NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

INTERACTIVE ENVIRONMENT FOR A
COMPUTER-AIDED DESIGN SYSTEM

by

Duard Stephen Woffinden

June 1984

Thesis Advisor: Alan A. Ross

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): Interactive Environment for a Computer-Aided Design System	5. TYPE OF REPORT & PERIOD COVERED: Master's Thesis; June 1984
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Duard Stephen Woffinden	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93943	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93943	12. REPORT DATE: June 1984
	13. NUMBER OF PAGES: 157
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Interactive Environment
Computer-Aided Design
Human-Computer Interface

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

There has been increased interest in the design of human-computer interfaces since personal computers have become popular. This thesis examines several existing

(20. ABSTRACT Continued)

design methodologies for creating a human-computer interface and then proposes a set of general design principles to be followed. An initial implementation of a human-computer interface for the computer-aided design system CSDE is presented.

Approved for Public Release, Distribution Unlimited.

An Interactive Environment for a Computer-Aided Design System

by

Duard Stephen Woffinden
Captain, United States Army
B.S.E.E., Utah State University, 1975

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1984

ABSTRACT

There has been increased interest in the design of human-computer interfaces since personal computers have become popular. This thesis examines several existing design methodologies for creating a human-computer interface and then proposes a set of general design principles to be followed. An initial implementation of a human-computer interface for the computer-aided design system CSDE is presented.

TABLE OF CONTENTS

I. INTRODUCTION

A. THE PROBLEM

The computer has evolved from a special purpose scientific tool into a general purpose tool serving a wide variety of users in many different applications. As the use of the computer has changed, the profile of the computer user has changed also. The technically oriented user, with a knowledge of the internal operation of the computer, and at least some proficiency in the computer's own language, has been replaced as the primary computer user by a large number of different individuals, each proficient in some specific field, but not having, or possibly even wanting, a detailed knowledge of computer science. Indeed, since the development of microcomputer technology, the use of computers has spread until most members of modern society interact with some type of computer almost every day no matter what their background or profession.

The change in the computer user's profile has brought increased attention to the interface between the user and the computer. The cryptic form of most computer commands has caused people with less technical backgrounds to view computers with emotions ranging from awe to fear. Experience indicates that when humans find a system inconvenient or difficult to use, they will avoid using the system,

even when the computer offers a significant increase in efficiency.

The financial incentives and intense competition in the personal computer market have brought many improvements to the human-computer interface. One idea that has developed from personal computer applications is the concept of the individual computer workstation tailored to the user's needs. This idea has special application in the area of computer-aided design, since it provides a useful and flexible tool for the individual designer.

However, since the design process often depends heavily on what is being designed, and because the variety of things that can be designed with the assistance of a computer is so broad, the problem is to create an environment which will be most conducive to the particular design process.

B. PREVIOUS WORK

Computer-aided design is not a particularly new concept. In some areas of industry, such as automobile and aircraft design, extensive computer-aided design systems have been successfully implemented. Sherlock [Ref. 3] gives an excellent overview of four computer-aided design systems as well as comments on the current situation in computer-aided design.

An application of computer-aided design to building real-time control systems was presented by Matelan [Ref. 1]

in 1976. Ross [Ref. 2] further refined the concepts presented by Matelan and implemented a system to perform computer-aided design of microprocessor-based controllers. Both of these presentations concentrate on the automation of the design process without specifying the form or the implementation of the "front-end," the human-computer interface.

The system implemented by Ross is known as the Computer System Design Environment or CSDE. This system operates on controller design specifications formulated in the syntax of a language known as the Control System Design Language or CSDL. CSDL was created by Matelan as a universal means of describing control system specifications. In the definition of CSDL, Matelan divided the design specification of a controller into four sections. These are the Identification Section, the Environment Section, the Contingency List, and the Procedures Section. Ross modified CSDL slightly and added the Design Criteria Section. Ross also described a primitive list format as the "compiled machine-code" for CSDE. However, without a front-end, the initial implementation of CSDE required that the designer first write the design description in CSDL syntax, and then hand-compile the description into the primitive list form required to run on the system. Hand compilation tends to be a tedious and error prone exercise for humans, while computers perform such tasks very well.

The first attempt at providing a front-end, human-computer interface for CSDE was presented by Sherlock [Ref. 3] in 1983. The interface is described as a user-friendly, syntax-directed input. Sherlock was able to specify the design of the input interface, however the graphics terminal available for the implementation proved difficult to control and totally unsatisfactory for this application. In fact, the problems were so overwhelming that the front-end could not be fully implemented. A description of the terminal and the problems encountered are detailed by Sherlock [Ref. 3].

The front-end implemented by Sherlock uses a modified form of CSDL called CSD/ADA. The interface is a menu-driven sequence of screen displays which guide the designer through a design in CSDE. Only syntactically correct CSD/ADA design specifications are accepted with error messages displayed if the user violates the syntax of the language.

As stated earlier, due to hardware interface problems, the front-end was not completely implemented. Several parts of the language such as the simple do, every, and at time options for contingencies are not supported. There is also no translation from the CSD/ADA form of the description into the primitive list format required by the existing implementation of CSDE. These missing parts are critical to the success of the input interface and emphasize the impact that the choice of display/input device can have on the implementation.

C. THIS PROJECT

The lessons learned from Sherlock's work led to dividing the front-end into two sub-areas. One area is the translation of controller specifications written in CSDL into the primitive list format required by the rest of the system. The realization of a solution for this sub-area is given by Carson [Ref. 4]. The remaining sub-area, which is the subject of this presentation, is the creation of an interactive interface which will allow the designer to concentrate on design procedure while the interface generates correct CSDL syntax.

Two important issues relating to the human-computer interface for the front-end of CSDE were not resolved in Sherlock's work. These issues are first, the choice of the type of display/input device on which to implement the interface, and second, the choice of a language for the human-computer interface. These were the initial issues that had to be resolved for this project.

Concerning the first issue, Sherlock's work clearly demonstrates the impact which a particular choice of hardware can have on the implementation. However, most often this choice is driven not by what is wanted, but instead by what is available. Fortunately, since Sherlock's initial efforts, two additional display/input systems became available. These are the Sun Workstation[1] and the AED 767[2] graphics

---

[1]Sun Workstation is a registered trademark of Sun Microsystems, Inc.

[2]AED 767 is a registered trademark of Advanced Electronics Design, Inc.

terminal. Both systems provide a bit-mapped display with a powerful set of graphics routines for creating screen images. One major difference between the two systems is that the Sun Workstation is designed as a stand alone system, while the AED 767 is designed to work from a host computer. This difference between these two systems actually became the determining factor in the choice of which system to use since both provide relatively easy to access graphics packages. The AED 767 was chosen as it could most easily be interfaced with the computer on which the rest of the CSDE system already exists. A future consideration may be to migrate the entire CSDE project to a system like the Sun Workstation.

The second issue, the choice of a language for the interface, has a great impact on the human involved with the system, as it will become the language the human user must know and understand to effectively use the system. Matelan [Ref. 1: pg. 106] presents a discussion, with additional references, on the impact of language on thought. Essentially, the most productive methods are those which avoid burdening the thought process with a need to translate between two or more languages. In other words, it would be best if the designer could enter design specifications into the computer in exactly the same language he would use without the computer. This is currently a very difficult problem as humans tend to deal in ambiguous languages which

are not well suited for computers, and also because every human tends to have a favorite personal style making it essentially impossible to please everyone. In spite of these obstacles, a language must be chosen for the interface. In the initial implementation of CSDE, the language of the human-computer interface was actually the primitive list format compiled manually from CSDL. Sherlock [Ref. 3] discusses several existing hardware design languages including CSDL and provides a justification for modifying CSDL into CSD/ADA. The resulting system requires the designer to learn CSD/ADA and the compilation into the primitive list format must still be done manually. From this it should be very clear that the choice of an interface language is a most important and difficult decision. This decision alone will have a great impact on the success or failure of the system. In making this decision for this project, two things were noted. First, since there now exists in CSDE a translator which converts from CSDL syntax into the primitive list format, provided by Carson [Ref. 4], and since this is to be the front-end for CSDE, it is most logical that the interface should generate correct CSDL syntax. Second, as mentioned previously, the goal should be to create an interface language which will allow the designer to formalize a controller design specification without having to learn CSDL. Therefore a translator for converting some more "natural" human form of controller design specification

format into the proper CSDL description must be developed. The "natural" human format is then the language of the interface. As already discussed, it is not anticipated that a totally "natural" human language which would please everyone could be found. Also as ambiguous languages are not currently feasible in computer applications, the interface will have to require that the user have some understanding of CSDL. Since the syntax of CSDL is rigidly defined, it can be embedded in the interface thus only requiring the designer to insure that the semantics of the design specification are expressed in the semantical constructs available in CSDL.

The presentation of the development of the input interface for CSDE proceeds from the foundation laid in this introduction. Chapter II presents background information that is necessary for the effective design of the human-computer interface and the interactive design environment. Chapter III details the assumptions and design decisions for this project. Chapter IV describes the implementation of a program to realize the design environment for the front-end interface. Finally, Chapter V provides a summary of conclusions and recommendations which resulted from this work.

II. BACKGROUND

A. THE HUMAN-COMPUTER INTERFACE

1. Theory and Issues

Interest in the human-computer interface is not new. In 1960, Licklider [Ref. 5] discussed the creation of a man-computer symbiosis. Allowing a loose definition of living and organism to include the computer, Licklider presents his ideas on what the interactive interface would need to be in order to achieve a symbiotic state between humans and computers. The areas which he feels must be addressed in the design of the interface include the division of function between the human and the computer, hardware selection to include input and output devices, and the language of the interface. Licklider's presentation represents a pioneering work in the study of the human-computer interface, however the presentation mainly deals in generalities and in the limitations imposed by the available technology of that time. It would seem that with the great advances in technology over the twenty-three years since Licklider's work, the design principles for creating a human-computer interface should be well established, if not already a "cut-and-dried" routine. However, such is not the case. One of the first things one notices in setting about to create a human-computer interface is the lack of

any concrete, structured procedure or method to guide the design process. This situation is best expressed by the following three quotes from recent literature on the subject. In 1978, Jones voiced the following opinion.

> ...I have come to the conclusion that we are only tinkering with the art of man-machine dialogue. Like a child learning to draw, we do the first thing that comes into our heads and are delighted with it just because it's pretty. We have no underlying principles, like the rules of perspective, to help us accurately convert an idea into reality and we have no set of standards, like those of proportion and composition, by which to criticize and improve our performance. And yet, in the very near future, man-machine dialogue is going to be really important. [Ref. 9: pg. 260]

More recently, this same theme has been echoed by Bailey (1982).

> Unfortunately, designers do not yet have a complete set of general design principles that, if followed, would result in the desired level of human performance. When confronted with human/computer problems many designers simply do the best they can. They have few meaningful human performance principles to help in accurately converting their ideas into reality, and, consequently have no set of standards by which to criticize and improve their performance. Most available information is in the form of opinions. [Ref. 7: pg. 293]

And also by Foley and Van Dam (1983).

> Like architecture, the design of user interfaces is at least partly an art rather than a science. We hope that the design of user interfaces will someday become more science than art, but the climb to reach this goal is long; the ascent has begun, but there are many hard traverses. [Ref. 6: pg. 217]

The progress in this area of computer science seems painfully slow, especially when compared with the rapid advances in other areas of computer related technology. Fortunately,

the situation is not as hopeless as it may appear for there are many successful systems which humans interface with very well, and many designers have shared their experience in recent literature. Thus perhaps the best way to begin the design of a new interface is to study the work of others and use their experience to advantage while avoiding at least the more obvious errors. The point to be made is that under the current conditions existing in the field of human-computer interface design, the designer is really on his own and must develop his own set of guidelines which he will follow, even though he may borrow from the experience of others. One caution is that in establishing a set of guidelines or goals for a design, it is often tempting to use catch phrases or buzz words to describe desired system attributes. Current human-computer interface design catch phrases and buzz words include user friendly, ergonomics, and user engineered. This can be a dangerous practice as these phrases and buzz words lack uniform definitions and thus do not provide a good means of communicating basic concepts.

Figure 1 gives an outline of general guidelines proposed by six sources. It should be noted that although different wording is used, there is a great deal of similarity in the concepts expressed. All six either specifically state or imply that the first principle is to know the user. A study of the rest of the principles listed reveals that

Hansen [Ref. 11: pg. 225] (1971)	Foley and Wallace [Ref. 8: pg. 25] (1974)	Jones [Ref. 9: pp. 262-265] (1978)	Bailey [Ref. 7: pp. 296-299] (1982)	Foley and Van Dam [Ref. 6: pg. 55] (1983)	Norman [Ref. 10: pg. 257] (1983)
Know the User	Know the User	Expectations of the User	Understanding the User	Simple Consistent Interaction Sequences	Design for Expected Human Error
Minimize Memorization Selection not entry Names not numbers Predictable Behavior Access System Info	Avoid Psychological Blocks Boredom Panic Frustration Confusion Discomfort	Predictability and Familiarity	Ensuring User Control	Do not overload the user with options and styles	Give feedback to the state of the system
Optimize Operation Rapid execution of common operations Display inertia Muscle Memory Reorganize Command Parameters		Include implication where possible (defaults)	Maintaining Consistency	Prompt novice but allow prompt bypass for experienced users	Keep similarity and differentiation in response sequences as appropriate
Engineer for Errors Good Error Messages Engineer out Common Errors Reversible Actions Redundancy Data structure integrity		Allow user Experimentation (Error Recovery)	Minimize Processing Time by People	Give Appropriate Feedback	Provide Reversible Actions
		Motivate User (Build Confidence)	Keeping User Informed	Allow Graceful Recovery from Mistakes	Be Consistent
		Consider Visual (Display) and Psychological (wait time, etc.) Issues	Improving User Accuracy and Facilitate Error Handling		
			Optimize Training		

Figure 1. Human-Computer Interface Design Principles

they all require some previous assumptions about or knowledge of the general characteristics which describe the intended user of the system. In the past, the human has been viewed as being adaptable and thus systems requiring a human-computer interface were designed with efficiency and ease of design for the system hardware and software, but with little regard for the human element. This is illustrated by the cryptic form used for many command languages which are efficiently handled by computers and are relatively easy to design, but which many humans find very difficult to learn and remember. Now that the human has become the most expensive component in many interactive computer systems, it has been realized that increasing the ease by which humans can communicate with computers will produce great benefits even though internal efficiency must be sacrificed.

None of the lists of general design guidelines studied were found to be completely satisfactory. Many seem to have become over concerned with the human issue and seem to fall short of giving guidance revelant to the complete design process for an interactive system. For example, the six lists shown in Figure 1 do not cover such issues as initial problem description and selection of the non-human components of the system. Martin [Ref. 12: pg. 10] presents the most nearly complete design procedure found. Since no one satisfactory set of guidelines was found, a proposed list applicable to this work is presented in Figure 2.

1. Determine the purpose of the system
2. Know the user
3. Identify resources available
4. Consider human factors
5. Determine the interface language
6. Consider the environment of operation
7. Design for evolution
8. Optimize training
9. Accommodate levels of experience
10. Use selection vs. entry
11. Be consistent
12. Anticipate errors

Figure 2. General Design Principles

It is difficult to convey the intended scope of each design principle in a short phrase, therefore a brief discussion of each principle is presented here. It should be noted that an attempt has been made to list each principle in the order of its consideration during the design process. Although some principles lend themselves to an order, it is really difficult, if not impossible, to establish an absolute sequence in which the principles are to be considered and applied to the design of an interface. Feedback must be considered as a continuous part of the design process which may alter earlier design decisions. The impact of feedback on the design may be kept to a minimum by careful application of each principle while continuously being aware of its interaction with other principles.

The first principle is determine the purpose of the system. The one overall goal for the application of this principle is to ensure that there is a clear understanding of the task the system must perform before decisions affecting the interface are made. This is of critical importance as the designer of the interface may not have any previous experience in the application for which the system will be used. Applying this principle should produce the foundation on which the rest of the design is built. This includes such things as a description of the system, description of the expected input and output, breakdown of tasks within the system, identification of the intended users of the system, anticipated conditions under which the system will operate, identification of existing components which are to be integrated into the design of the interface, and any other high level considerations the designer feels are important.

Know the user is the second principle because the identification of the intended user is a part of the initial system description. This is perhaps the most difficult principle to apply. The goal of this step is to gain an understanding of the personality of the intended users as a group. Items which should be considered include level of education, social background, prior use of computer systems, types of job related dialogue used, and any other general characteristics of the intended users which may relate to

their interfacing with the system to be designed. It is important at this point to get as much user input as possible, to make the user a partner in the design process. This may be accomplished through the use of surveys, interviews, and so forth. This principle should be applied throughout the rest of the design process. There are situations where it may not be feasible or even possible for the designer to receive user input before the design is completed. In this circumstance, the designer should at a minimum make a careful list of the assumptions he is making about the general characteristics of the intended users.

Next is identifying resources available. The results of this step will usually establish limits as to which features can be implemented. The choice of the hardware and software to be used in the implementation will have a great impact on the interface that is achieved as was demonstrated in Sherlock's work. If a choice is available, the items of importance include types of display devices such as CRT's and printers, image refresh and update speeds, graphics features such as colors available, text size and style, and drawing capabilities, available memory, as well as software capabilities such as the available programming languages, graphics commands, and operating systems.

The human factors to be considered are divded into two categories. The first category is physical factors. These are such factors as color perception, manual dexterity,

comfort of the human body, eye strain, color coordination, physical handicaps among users, and so on. The physical factors actually accommodated in the implementation will be limited at least in part by the resources identified in the previous step. The second and much more complicated category is psychological factors. These factors are composed of the multiple dimensions of human nature. As many psychological factors as possible should be considered and included in the design. Some important factors which should be considered include psychological blocks such as boredom, panic, frustration, and so on, achieving psychological closure, keeping the user informed of the state of the system, reassuring the user that he is in control, minimizing memorization, determining appropriate response times, and insuring the user is not overloaded with options or styles. It is easy to become bogged down in a myriad of psychological issues. This may be avoided by using the information gained from applying the first three principles to establish limits on the design and thus identify the psychological factors to be considered. Excellent references relating to human factors are Bailey [Ref. 7], Martin [Ref. 12], and Card, Moran, and Newell [Ref. 13].

The first four principles should help to establish some limits which may be used in defining the language of the interface. As mentioned in the Introduction, the most desirable choice would be to adopt the current language used

to accomplish the tasks for which the interactive computer system is being designed, but as also noted earlier, this is not usually possible due to ambiguities in human languages and lack of standardization. The goal is then to come as close as possible avoiding making the user learn a new and unfamiliar language. It should be remembered that the language of the interface is the mechanism by which most of the psychological needs are fulfilled.

The sixth principle given here is to consider the environment of operation. This includes the physical layout of the work area, and such intangibles as the atmosphere of the work environment. The application of this principle is implicitly included in at least the first four principles. However, it is often not addressed explicitly and thus it has been included for completeness. The types of issues considered are mostly human comfort and efficiency issues such as the effect of lighting on a screen display, and so on.

Design for evolution is an important principle in the design of interactive systems. It is not likely that the designer will be able to identify all of the desirable features which could be implemented in the system. It is also unlikely that the designer will be able to predict the impact the new system will have on the environment in which it is to be used and what changes will thus result. Therefore, the system must be designed with an ability to evolve. This is discussed in more detail in Section B.1 of this chapter.

The eighth and ninth principles, optimize training and accommodate levels of experience, are inter-related. The goal of optimizing training is to allow a new user to be able to accomplish meaningful work on the system without the assistance of a more experienced user. The goal should also include having the capability built into the system to provide refresher training for more experienced users as necessary. It should be obvious that the amount of training provided and the method of access to the training should be a function of the level of experience of the user. Various numbers of levels have been recommended. Sherlock [Ref. 3] presents five levels of experience, however, three levels appear to provide a minimum set. These are the novice user, the casual user, and the experienced user. An example of each type of user for a system would be a new employee for novice user, a supervisor for casual user, a long term employee for experienced user. Each of these classes of user requires different levels of refresher training. A detailed discussion of interactive and embedded training mechanisms is presented by Groenert [Ref. 15].

The next principle is to use selection versus entry. This means it is preferable to allow the user to pick an option from some sort of list rather than requiring the user to enter a choice through spelling out commands with keystrokes. A verification of the importance of this principle may be observed in the popularity and success

of certain pick devices such as the mouse in personal computer systems.

Throughout the design, it is very important to be consistent. This refers to the commands used as well as their format. For example, the procedure for accessing the help function, for saving work, and for exiting the system should be the same in all levels of the system.

One of the greatest assets provided by computers is the ability of the computer to remember and to inter-relate details. It is this capability which should be exploited when applying the principle of anticipating errors. Norman [Ref. 10] provides a discussion of the types of human error which may be anticipated. These include mistakes and slips which are further refined into mode errors, description errors, capture errors, and activation errors. Many of these errors result from decisions made in specifying the language of the interface, therefore this principle should be applied in conjunction with the determining of that language. One large class of errors is related to the syntax of the language. A means of eliminating this type of error is to embed the syntax of the language in the system. This is the method employed by syntax directed editors which are finding wide acceptance today.

Once the general design principles have been established, the designer can begin to apply them to the design of the interface. At this point, the interactive interface

should be viewed as having two components, the physical interface, and the psychological or language interface. As each principle is applied to the design process, both of these components should be addressed. Some desired features of the system will fit neatly and logically into one category or the other while other features will fit into some combination of the two categories. Tradeoffs between the two categories will be required and must be carefully documented as the design process progresses.

2. The Physical Interface

The scope of the physical interface, as its name implies, includes the more tangible physical attributes of the system. Issues relating to this interface can be further divided into the attributes of the human user and the capabilities of the computer equipment.

The human side of the physical interface deals with physical characteristics of human beings such as manual dexterity, color perception, hand-eye coordination, and body dimensions. Also of interest are types of physical handicaps which may be encountered such as color blindness, deafness, and so forth. Much of the concern over the physical interface from the human side relates to the ease and comfort with which a human user may interface with the system. Most of the work concerning human physical characteristics relates to more tightly coupled human-machine interfaces such as the interface between a pilot and the aircraft he flies. In

designing such man-machine interfaces, human performance engineering has become an important part of the design process. In more loosely coupled applications such as the human-computer interface, the physical performance of the human appears to have much less impact than for tightly coupled systems and thus human engineering is often left out of the design process. However, good design procedure encourages the use of all available means to help produce the best result, therefore the human physical characteristics should be considered. For example, Professor George A. Rahe of the Naval Postgraduate School in teaching the "Interactive Computer Graphics" course, observed that if a computer system included a visual output device such as a CRT screen, then it must be assumed that there would be a set of human eyes intended to view that output, and thus the characteristics and limitations of the human eye should be considered as part of the design of the use of that output. An excellent resource which presents both the human physical characteristics for the more general human-machine interface as well as the human-computer interface is Bailey [Ref. 7]. The human physical characteristics which are important for the design of a given system should be known once the first two design principles have been applied.

The computer side of the physical interface deals with the capabilities of the computer system hardware. The capabilities of interest include such things as whether or

not color is available and if so, what color combinations may be displayed, the layout of the keyboard if it is to be used for input, whether or not special function keys are available, and what other devices such as light pens, touch screens, a mouse, etc., are available. The choice of which devices should be used needs to be based on the user profile previously generated. Such things as user preference and the average typing skills of the user are some considerations. In addition to what types of components are available, such things as the system construction are also important. This relates to whether the screen, keyboard, and computer are all physically connected together, or if they may be moved separately. The capabilities of the hardware should be identified after applying the first and third of the general design principles shown in Figure 2.

Once the human physical attributes of importance and the computer capabilities available have been determined, the creation of the physical interface involves carefully matching the two together to achieve the best interface possible. This matching of the human side with the computer side may be a difficult process. First, it may be difficult to completely determine all of the physical characteristics of the user group along with the relative importance of each characteristic. Many such difficulties will relate to human physical handicaps and determining their impact on the design. An example of this would be determining if color

blindness is a critical factor and if so, how much of a factor it is. Again, the designer must carefully document assumptions made to help accommodate future modifications which are almost certain to be necessary. Second, as was noted in the Introduction, many times the designer is limited as to the hardware available to implement the system. Obviously the designer has much greater flexibility if new computer hardware is being created as part of the system, than if he is constrained to use already existing hardware with predetermined capabilities. Under the latter conditions, the designer should note any significant problems encountered to be of future assistance when new equipment becomes available. The physical interface is realized when the matching of attributes with capabilities is completed. It should be noted that this is actually the "easy" part of completing the overall human-computer interface as humans are adaptable creatures who will put up with some inconvenience or discomfort as long as they feel the benefits received are great enough.

3. The Language Interface

Whereas the physical interface deals with the more tangible physical attributes of the human-computer interface, the language interface deals with the less tangible attributes of the human mind and human psychology. It is this part of the human-computer interface which tends to receive the majority of attention. A detailed discussion of this

extremely complicated topic is beyond the scope of this work. Therefore the reader is referred to such works as Foley and Van Dam [Ref. 6], Bailey [Ref. 7], Martin [Ref. 12], and Card, Moran, and Newell [Ref. 13] for further information on this subject. What is presented here is a brief look at some of the important aspects of the language interface and how this may be applied to the creation of the interface as a whole.

A common way to approach the design of the language interface is to view the interface as if it were a dialogue between the human and the computer. Foley and Wallace [Ref. 8] and Jones [Ref. 9] each propose methods of designing human-computer dialogue by drawing parallels with human to human dialogues. One major difficulty with this type of approach is that it is hard to capture the full essence of human to human dialogue due to the use of such non-verbal communications as "body language," voice inflections, and implied but unspoken meanings. The result is that the designer must either find or create some more rigidly specified language which is better suited to computer applications. It should be noted that certain choices for the language interface may also impact on the physical interface further complicating matters.

Some of the important topics to be considered in creating the language interface are those previously presented in the general design guidelines under the psychological

issues relating to the fourth principle, Consider Human Factors. Also revelant are the issues covered in the discussion of principles five through eleven. Some commonly accepted practices such as limiting choices to no more than seven are related to user short term memory. Another important consideration is providing for psychological closure, or in other words, fulfilling user expectations. Providing psychological closure is well illustrated by examples from telephone service. These examples are presented by Sherlock [Ref. 3] and Martin [Ref. 12].

The success of the language interface will depend largely on how well the first two design principles are applied to the design. A thorough description of the purpose of the system and a careful analysis of the user group for the system will provide the basis required to specify an acceptable language. This language must include two capabilities. The first is the capability to express administrative instructions for use within the system, and second, the capability to express instructions relevant to accomplishing the purpose for which the system is intended. In the case of this project, the first capability would have to accomplish entry to the system and handling of design files, whereas the second capability would have to deal with how controller designs are specified for further processing within the system.

Perhaps the most effective way to put all of this into perspective is to remember that one of the primary goals of the interactive human-computer interface, as was stated in the Introduction, is to allow the human to communicate his ideas in the same language he would use without a computer. In other words, if at all possible, not to burden the user with the need to learn a new language.

An observation which may be drawn from the study of natural human languages is that they tend to be dynamic, constantly changing. Likewise, computers are continuously changing and increasing in capability. Therefore, the combination of the language interface with the physical interface to create a human-computer interface should be viewed as an evolving process always seeking to increase the bandwidth of communication between the human and the computer. This is the driving force behind the recent interest in the creation of interactive programming and design environments.

B. THE INTERACTIVE ENVIRONMENT

1. Theory and Issues

It may be argued that the "terms," "human-computer interface" and "interactive environment," really describe the same thing, so why devote separate sections to their discussion. This work contends that those systems described as being interactive environments are really a special subset

of the universe of human-computer interfaces. The purpose of this section of the chapter is to examine the requirements which qualify an interface to be considered an interactive environment and to determine what impact those requirements have on the general design principles previously proposed.

Perhaps the most significant characteristic common to interactive environments is the creation by the environment of an allusion to or metaphor of something familiar to the user. An example of this technique is the use of graphics "windows" to create an allusion to a desk top covered with papers. The goal of the interactive environment is to use a symbolic representation of things the user is already familiar with to create in the user's mind the illusion that he is dealing with something more than just a computer system. Winograd stated such a goal in his conception of a programming tool.

> We can better think of it [the tool] as a moderately stupid assistant, to whom we give all the information we possibly can, and who in turn relieves us of much of the burden of memory, tedious checking, and drawing more or less straightforward conclusions. [Ref: 16: pg. 5]

The level of intelligence of the tool is not the most important issue. Of much greater importance is the idea of having the designer of the tool create that tool from the conceptual view of an assistant to the programmer, with the goal being to have the user of the system view the result not as a computer system, but instead as an assistant. This

view of the interactive environment is also illustrated by an analogy related by Professor Daniel Davis of the Naval Postgraduate School which he had heard presented at a conference he had attended. The analogy concerned the similarities between what a motion picture director and the designer of an interactive environment are trying to achieve. The motion picture director is trying to get a user group to become involved with the story line or plot of the movie without thinking about the film, the projector, and two dimensional screen on which the sequence of pictures is presented. In the same sort of way, the designer of an interactive environment is trying to get a user group to become involved with a symbolic representation of a desired environment without thinking about the computer system on which the environment is implemented.

Another characteristic of interactive environments is that it is extremely difficult to generate specifications for them. They are usually very complicated and programs which implement them tend to be very large. The following quote from Sheil expresses this diffulcty.

> The implementation disasters of the 1960's however are slowly being succeeded by the design disasters of the 1980's. The projects...simply do not yield to conventional methods. Any attempt to obtain an exact specification from a client is bound to fail because as we have seen, the client does not know and cannot anticipate exactly what is required....The alternative to this kind of predictable disaster is not to abandon structured design for programming projects that are, or can be, well defined. That would be a tremendous step backward. Instead, we should recognize that some applications are best thought of as design problems rather than implementation projects. [Ref. 17: pp. 20,22]

The characteristic of being difficult to specify is closely related to a third characteristic of the interactive environment. This third characteristic is that they are dynamic in nature. The objects and actions for which the interactive environment creates a symbolic representation are constantly changing. This constant changing requires that the interactive environment be able to evolve. Concerning the evolution of programs, Lehman [Ref. 14] discusses three classes of programs in the context of their design, use, and maintenance. The interactive environment being discussed here falls into the class of programs referred to by Lehman as E-programs. These are programs for which events and objects in the environment within which the program is to be embedded, have some effect on the design, use, and maintenance of the program. Lehman notes that it is usually not possible to predict how the environment will change once the program is introduced into that environment. If the changes are of any significance, Lehman contends that the program must be able to evolve to fit the new environment or the program will become increasingly inefficient and soon will be obsolete. This is especially true for interactive environments which must be able to change and adapt to new needs and expectations of the user.

Interactive environments are thus viewed as a special subset of human-computer interfaces in general. They are characterized as creating an allusion to something

the user is familiar with, being extremely hard to generate specifications for, and being dynamic in nature. It is therefore necessary to review the general design principles and methodologies previously discussed to see if they may be applied to the design of an interactive environment.

2. Design Principles

The general design principles listed in Figure 2 are general in nature and may be applied to the design of any human-computer interface. Given a specific interface or class of interfaces such as the interactive environments, it is necessary to define the scope of various principles to meet the characteristics of the interface to be designed. In the case of the interactive environment, it is proposed that the principles most affected are the first, second, fifth, and sixth as discussed next. Of course the impact on these principles will have repercussions through the rest of the principles as they are often inter-related. However, these four principles can be used to establish guidance for the rest of the design procedure.

The first principle is determine the purpose of the system. Since interactive environments are difficult to generate specifications for, the designer must be very flexible and prepared to accept changes. This is very distasteful to those with a strong belief in structured programming principles, yet such an approach has proved very successful for such interactive environments as InterLisp.

Sheil cautions those who would impose structure techniques on all applications with the following observation.

> Those who admire the massive, rigid bone structures of dinosaurs should remember that jellyfish still enjoy their very secure ecological niche. [Ref. 17: pg. 30]

This means that the designer should accept the fact that it is unlikely that all important considerations will be realized to begin with and therefore careful documentation of all assumptions and designs to accommodate evolution must be part of the plan from the very beginning.

The impact on the second principle, Know the User, is very similar to the impact on the first principle. The needs and expectations of the user will change. Therefore the designer must carefully document his choices, design for evolution, and accept the fact that not everyone will be pleased with the result. The amount of difficulty experienced due to the users needs and expectations can usually be decreased dramatically if representatives of the user population can be made partners in the design process.

The fifth principle is to determine the interface language. For the designer of an interactive environment, the goal here should be to develop a set of primitive building blocks which may be combined to create new features in the language as each new feature is identified. Such an approach has proven very successful in interactive environments such as INTERLISP. An excellent collection of papers

which present many good examples of this approach is Barstow, Shrobe, and Sandewall [Ref. 18].

The sixth principle, Consider the Environment of Operation, is included as a reminder to the designer that the environment will change often in unpredictable ways. Perhaps the most important thing which the designer who creates an initial implementation of an interactive environment must keep in mind is that he should design the system to be able to evolve as easily as possible.

Now all that remains is to apply the design principles in the context of interactive environments to a specific situation. It is in a specific application where the procedures will begin to take form. The application of this background to the creation of an interactive environment for CSDE is the subject of the next chapter.

III. DESIGN CONSIDERATIONS

A. DESIGN PHILOSOPHY

An early part of the design process should be the adoption of a set of principles or guidelines to be followed, and then the definition of a philosophy as to how those principles or guidelines are to be applied. It would be hypocritical to have presented a proposed list of acceptable principles as was done in the previous chapter, and then not to apply those same principles to the design of the project at hand. Given that those twelve principles are to be applied, the proposed design philosophy is to be as specific as possible early in the design process, avoiding the practice of delaying design decisions. Though this is the opposite of the more traditional view that binding decisions should be made as late as possible, it is argued here that for the design of a human-computer interface, particularly an interactive environment, it is better to limit the scope of the design, especially the initial implementation, and to handle the need for change by building flexibility and extensibility into the environment. This should help avoid the common design tendency of trying to create a "Swiss Army Knife" type of system that seeks to be everything to everyone. Even though generality is to some extent desirable, it is proposed that the design process should deal with specifics

first and handle any need for generality through designing for evolution.

In keeping with this philosophy, it is noted that the twelve design principles may be viewed as being composed of two groups. In one group are those principles which can be referred to as "what" principles and in the other group are those principles which can be referred to as "how" principles. To illustrate this viewpoint, the twelve principles are restated in Figure 3.

1. What is the Purpose of the System
2. What is the User Like
3. What Resources are Available
4. What Human Factors are Important
5. What is the Interface Language
6. What is the Environment of Operation
7. How to Design for Evolution
8. How to Optimize Training
9. How to Accommodate Levels of Expertise
10. How to Use Selection Vs. Entry
11. How to Be Consistent
12. How to Anticipate Errors

Figure 3. Restated Design Principles

Taking this view of the design principles will allow the designer to establish limits for the given design problem by being specific in the application of the first six principles, while flexibility for future change may be included

during the application of the last six principles. It is in this context that these principles are now applied to the design of an interactive environment for CSDE.

B. APPLICATION OF DESIGN PRINCIPLES

1. Determine the Purpose of the System

Simply stated, the purpose of CSDE is to determine the hardware components required and to generate the software necessary to realize an instance of a real-time, microprocessor-based controller from a set of specifications provided by a human designer. The focus of this work is on the interface between the human designer and CSDE. Therefore the purpose of the system to be designed here, an interactive environment to be the front-end for CSDE, is to translate human generated controller specifications into the correct CSDL syntax.

The high level considerations discussed in the previous chapter can now be addressed with specific information appropriate to an interface for CSDE. The target user group is composed of control system engineers. The flow of information, generation of input and output, and a division of tasks can be demonstrated by the following sequence. The design engineer has an idea or receives a request for a control system. From this the design engineer generates a specification for the controller in the interface language. The front-end of CSDE translates the interface language specification into the correct CSDL syntax which is passed to the rest of CSDE. Another important consideration to be addressed at this point

is that since the input interface is to be integrated into the existing CSDE system, the design of this interface must be restricted to implementations which will work with the existing system.

2. Know the User

The target user group was identified during the application of the first principle as control system engineers. Although this author and others involved with CSDE have some background in the design of control systems, there was no one available who possessed current experience in industry who could act as a partner in the design of this interface. Due to a number of restrictions, most of which were related to available time, it was not feasible to conduct a survey of engineers in industry. The decision was made to proceed with the design by listing assumptions about the user as was suggested in Chapter II. An important part of the evolution of this interface should be future work to poll members of the target user group and use the results of that poll to validate assumptions made here, or to modify the interface to more closely meet the user's needs.

The assumptions which are made about the target user group are as follows. The user is assumed to have received a college degree and to have some familiarity with computers in general. The user knows how to design control systems and has the ability to learn to use CSDE and CSDL. It is assumed that the user group has few physical limitations and

that the users have the ability to use the computer peripheral devices, such as the CRT, keyboard, and pick devices, which are currently available, however the user is not necessarily assumed to be a skilled typist. Perhaps most importantly, it is assumed that the user will be motivated to learn and use CSDE once its benefits are demonstrated.

3. Identify Resources Available

The primary resource to be used in implementing an interactive environment for CSDE is the AED 767 graphics terminal. The reasoning behind this choice was presented in the introduction. The existing components of CSDE are running on a VAX[3] 11-780 computer under the VMS[3] operating system. The AED 767 was designed to interface most easily with a host computer running the Unix[4] operating system. Part of the difficulties encountered by Sherlock [Ref. 3] were related to the Pascal compiler on the VMS system. Therefore, the decision was made to develop the environment for CSDE in the programming language, C, and run it under the Unix operation system. This was determined to be acceptable for a first implementation as CSDE currently uses files to pass data from one component to another. Unix is also well suited to interactive applications and it is easy to embed Unix system functions in programs written in C.

---

[3]VAX and VMS are registered trademarks of Digital Equipment Corporation.

[4]Unix is a registered trademark of Bell Laboratories.

4. Consider Human Factors

Without having direct input from members of the target user group, it is difficult to be very specific in this area. Given the assumptions made about the intended users of the system, the physical factors have few limitations. The assumption that the user has few if any physical handicaps and is able to use computer peripheral devices currently available, indicates the full capability of the AED 767 may be used as appropriate. However, because the user is not assumed to be a skilled typist, any use of the keyboard that emphasizes typing skills should be avoided. A physical factor which is of importance is the capability of the human eye. The capability of the eye will impact the layout of the screen and the choice of color combinations. The AED 767 is advertised as being able to display 16.8 million possible colors of which 256 are simultaneously available in the color table. This provides far more possible color variations than the human eye can distinguish, but it also means that there is plenty of flexibility to choose good color combinations. The graphics commands coupled with the bit-mapped display allowing each pixel to be individually painted if needed, makes modification of the screen display reasonably easy.

The psychological factors are extremely difficult to address without user interaction. Sherlock's [Ref. 3] discussion of user psychology was used as a guide for applying

this principle. Consideration of the human short term memory is handled by limiting choices at any one time to seven or less as was previously mentioned. Also, titles for display screens and appropriate prompts are provided to reassure the user that he is in control. It is not anticipated that all human psychological needs will be fulfilled by the initial implementation, therefore, a major design consideration is the organization of the structure of the interface to allow future modification.

5. Determine the Interface Language

The overall scope of this work is to design and implement a frontend for CSDE. As was noted in the Introduction, this limits the underlying language for the interface to CSDL. In addition to CSDL, the interface language also needs to be able to express administrative actions such as the creation, manipulation, and deletion of design files, the identification of users, and so on. This will be accomplished using the Unix Command language capabilities and shell facilities. With Unix and C language facilities to handle the administrative part of the interface language, CSDL is left to handle the controller specifications. Unfortunately, CSDL is not a uniformly accepted language for designing controllers, but there is no standard language for this purpose. If there were some standard, this interface would only have to translate from the standard into CSDL. The result of this situation is that the user of CSDE will have to gain

some understanding of CSDL. The intent of this design is to assist the user in knowing which semantic constructs of CSDL are legal at any given time while not requiring the user to know the syntax of the language.

6. Consider the Environment of Operation

The environment of operation will mainly impact user comfort. The environment is assumed to be either an office or a laboratory, and it is assumed that positioning of equipment, lighting, and noise level can be adjusted for the user. Therefore, for this particular application, the environment of operation will not be a major consideration.

7. Design for Evolution

The application of this design principle begins the decisions on how to implement the interactive environment. The presentation of the first six principles indicates that modification will be inevitable. The purpose of designing for evolution is to lessen the impact of incorporating needed modifications into the system. It is in the decisions made here that feedback in the design process and changes in earlier choices are accommodated. The designer must anticipate the future of the system and design to aid in the systems adaptation. In the case of CSDE, it is anticipated that the system will eventually be migrated to some individual workstation like the Sun Workstation. The popularity of the Unix operating system for work station applications led to the decision to base the implementation of the

environment for CSDE, in Unix and C. A major consideration for evolution is handling potential changes in system hardware. There is no standard graphics core that would insure portability of graphics routines, so the best solution found is to isolate graphics dependent functions from the structure of the system as much as possible. The C programming language allows separately formed procedures to be called from a main controlling program. This modularity will be used in an attempt to minimize the impact any modification will have on the overall system.

8. Optimize Training

One of the biggest drawbacks of many systems is that when the system is first brought in or when a new employee arrives, an extensive training program is required to teach the potential users how the system works. Ideally, the system should be designed to lead a new user through meaningful work without requiring the assistance of an experienced user. This will be done in this system through the use of prompts, and a help facility.

9. Accommodate Levels of Experience

An earlier discussion noted that a user changes with experience. The environment is to be designed with a user selected switch that will allow the user to choose between three versions of the environment, a novice version, a casual user version, and a version for the experienced user. The display and the order or progression through the environment

are the same for all levels. The difference between levels of expertise will be in the amount of prompting the system provides the user and the terseness of the commands to be entered.

10. Use Selection Vs. Entry

The application of this principle to this project lead to the decision to make a menu driven environment similar to the one Sherlock [Ref. 3] implemented. The system will prompt the user as to which semantic constructs are legal and allow the user to select the option he desires while limiting required entries to variable names and parameters needed to specify the function of the control system to be designed.

11. Be Consistent

This principle relates to the commands which will cause the system to function. It is important that these commands do not change as the user changes experience levels or as the user moves through a session. For example, in all parts of a secession, the command to quit the session should be the same. Also, the representation of a command in different levels of experience should all be related. An example of this would be to require a novice user to type "quit" while an experienced user would be allowed to use an abbreviation like "q" to quit. The commands needed for the operation of this environment are quit, save, create, edit, help, and delete.

12. Anticipate Errors

Most of the errors of a critical nature in this application deal with the actual controller specifications. It is not possible for the system to detect errors in the controller specification which deal with the conceptual design. However, the system should detect such things as undeclared variables that may be due to omission or misspelling. The handling of this type of error can be accomplished by cross referencing the user defined names used in the contingencies and procedures with those declared in the environment section, and prompting the user when the name has not been defined. The user should be able to re-enter the name if the error was in spelling or define the name and have it included in the design without the user having to leave the part of the interface he is working in. This will be accomplished by creating a table of identifiers which are entered in the environment section and then checking subsequent usage of identifiers with this list. Future error handling mechanisms should provide an undo facility which would allow the user to undo the effects of a previous command.

Syntax errors will be avoided by having the system include the required key words and punctuation required by CSDL and only allowing the user to make choices from options which are legal in the syntax of CSDL at that point in the controller specification. It is not considered desirable

to display the syntax of CSDL to the user while the specification is being entered as is done in most common syntax-directed editors. Instead, the user will be given the legal options as specified by the syntax with the formal syntax being available within the help facility. The idea here is that the user may wish to see the formal syntax of some part of CSDL and he should be allowed to do so, but the user should also be able to progress through a session without referring to the formal syntax.

C. DESIGN SUMMARY

The preceding application of the design principles has defined the limits for an initial implementation of an interactive environment for CSDE. The attributes of the system to be implemented may be symmarized as follows. The system is to be a menu-driven sequence of graphics displays which will guide design engineers through the specification of a controller design in CSDL. It is to be implemented using the C programming language on a VAX 11-780 host computer running the Unix operating system. The AED 767 graphics terminal is to be the user interface display device. The user will have to understand the semantic constructs of CSDL, but the syntax of the language will be handled by the system. Modularity will be used to isolate graphics dependency and to help reduce the impact of modifications. Three levels of user experience, novice, casual, and experienced, will be accommodated, and the system will

provide prompts and help to assist the user in learning the system without human assistance. Spelling errors and undeclared identifiers will be handled by cross-referencing user entered identifiers throughout the design session. This chapter has outlined the design of the interactive environment for CSDE, which is the easy part of creating an interactive environment. The challenge now is to implement this design with hardware and software. The specifics of this implementation effort are covered in the next chapter.

IV. IMPLEMENTATION

A. DATA STRUCTURES

One of the first decisions required in implementing an interactive environment for CSDE was how to handle user input data along with the syntactic keywords and punctuation required by CSDL within the interface program. A careful analysis of the grammar for CSDL, included as Appendix A, was made to help in determining how this data manipulation should be implemented. This analysis of CSDL revealed that sixteen of the nonterminals in the grammar could potentially be empty with two of the sixteen allowing for options of zero or more instances of their subexpressions at any one time. In addition, twenty-one of the remaining nonterminals allow options of one or more instances of their subexpressions at any one time. This means that whenever any of these thirty-seven nonterminals are encountered during an interactive session, the user must be given the option of repeating that part of the specification an arbitrary number of times. This, combined with the options for picking terminal symbols or strings in other parts of the grammar, creates a very difficult problem in choosing data structures.

Since this is the initial implementation of the CSDE generator, the decision was made to write the user entered data and CSDL syntactic constructs to a file as they were

generated during a design session. This is less flexible than creating data structures in memory, but considering the limited manpower available to work on this implementation, and the time constraints involved, it was decided that this provided the most expeditious means of providing a foundation for future work on the interface. One possible implementation of an editing capability would be to create a function to read a design file after it has been created by this initial implementation and then manipulate the data in memory.

B. ORGANIZATION OF PROGRAM MODULES

Once the decision had been made on how to handle design data, attention was turned to partitioning the implementation program into sub-modules. The first partition established was between functions for creating a design specification and administrative and hardware oriented functions. The implementation code is contained in Appendix B and is used to illustrate these different types of functions. The C programming language requires a main program be identified which may then call on external functions which are made accessible to it. The main program contains the core of the administrative control. The design specification functions include such external functions as "id-section" and "criteria-section" which generate and control the design sequence based on the user's choice. The design specification is broken into five parts, one for each of the design sections in CSDL. Each of these five was partitioned into a section

text function, menu functions, and a control function. For each of the five major areas, the identification section, design criteria section, environment section, contingency list section, and procedures section, there are a series of nested functions which lead the designer through the options available in the language. Finally, the hardware oriented functions include routines which initialize the AED 767 and set up the color table and graphics command formats, and routines like "find line" which moves the cursor to various positions on the screen.

After the program had been partitioned into the three major divisions discussed above, creation of C programming language functions to perform the required actions was begun. It was in the coding of the functions for the design specification module that the most difficulty was encountered. The recursive nature of CSDL combined with the number of options at any given level caused the program to grow very rapidly. Time constraints then forced the decision to make this initial implementation a skeleton on which future work could be added. A simple illustration of the complexity of the implementation is in the environment section where the designer has the options of leaving the section empty, or recursively entering any combination of input, output, duplex, binary, arithmetic, code, or code variable specifications an arbitrary number of times. The implementation of this structure was accomplished by creating nested levels

within which the program cycles through a loop. The designer is given the option of bypassing any unneeded areas, or staying in one area and entering an arbitrary number of specifications. Exit from any of the loops is accomplished by the designer choosing the "finished" option. This structure meets the requirements of CSDL, but requires a great deal of code. Almost two thousand lines of code are necessary for the environment section alone. This sounds large but is not excessive for interactive programs. Time limits, however, forced the decision to create program stubs for the arithmetic and binary environment specifications and for the contingency list and procedures section.

C. SCREEN DISPLAY

The choice of screen display format was influenced by the desire to be consistent and to keep the user feeling in control. This is accomplished by dividing the screen into four areas as illustrated in Figure 4. The title area contains a descriptive heading which informs the user about which design level is currently being occupied. The text/work area is used to inform the user about the design level being considered, and this is where user entered data appears on the screen. The menu area contains the options currently open to the user. Finally, the prompt area is used to display instructions to the user on the administrative control of the design process.

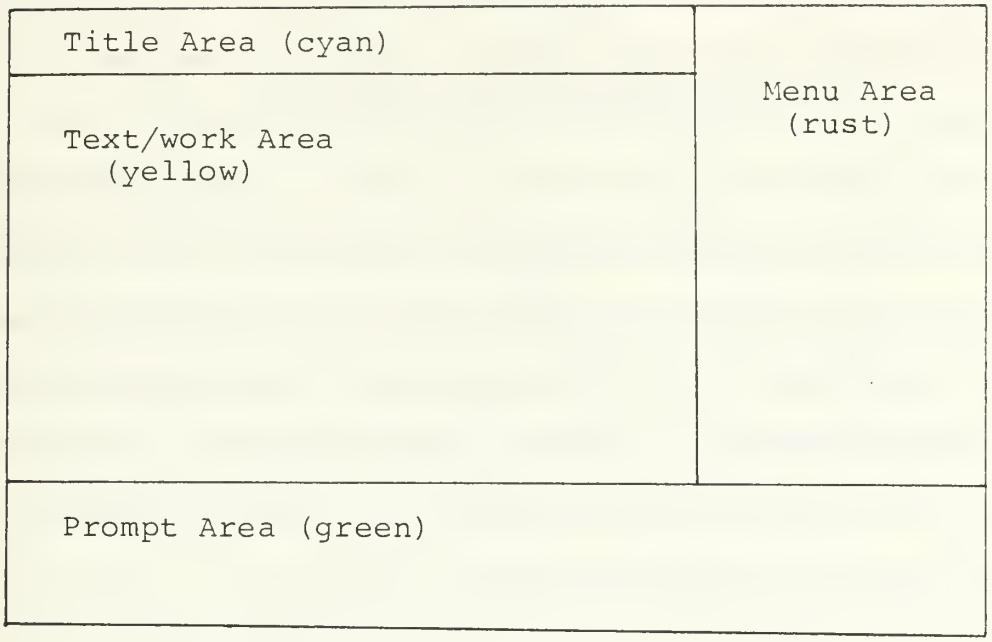

Figure 4. Screen Layout

D. OPERATIONS OF THE ENVIRONMENT

The best way to illustrate the operation of this implementation is to step through an abbreviated design session. The designer is first presented with a "welcome" screen from which he has the option of quitting or starting a design session. As he continues, the next screen is the system initialization screen. This is currently a stub screen which just presents the user with the option of leaving the program or continuing. This is where the choice of level experience and multi-user facilities can be added in future evolution of the interface. Next the designer enters the identification section where his name, the date, and the project title are requested for documentation. It should be

noted that the grammar for CSDL allows any or all of the major sections to be left empty. Once the designer enters a section, he is presented with a menu of options which will lead him through the entry of design specifications for that section. Once the session is completed, the designer is prompted to quit the system and is then informed that the design specification in CSDL syntax is contained in the file "output.csde." At the end of the program, the current color table values are displayed as an aid for future modifications of the interface.

E. ERROR HANDLING

This is another area which time did not allow to be treated completely for this initial implementation. The error handling implemented thus far is limited to checking the validity of the options selected by the designer. In the modules which have been implemented, human syntax errors have been eliminated as the proper syntactic keywords and punctuation are inserted by the program and not by the designer. For this initial implementation, the user should be warned that the error checking capabilities are very unsophisticated and failure to follow the printed instructions may cause failure of the program.

V. CONCLUSIONS AND RECOMMENDATIONS

A. ACCOMPLISHMENTS

Earlier in this work it was noted that there exists little in the way of concrete guidance for the design of human-computer interfaces. Therefore, one of the main efforts of this work has been to establish a set of guidelines and principles which will aid those who continue with this work in the future. It is hoped that the general principles have been thoroughly enough covered that future effort can be concentrated on implementation aspects of the system based on the background presented here.

Another accomplishment of this work is the evaluation of hardware new to the system. The AED 767 is an excellent graphics device, however, whenever a new device is introduced, there is a period of adjustment during which the capabilities of the device must be explored. Annotations were made to the Naval Postgraduate School's copy of the AED user's manual [Ref. 19] as problems were encountered, to aid future users of the system. Also, several programs which demonstrate various capabilities of the AED were created and these are left to aid those who follow on.

Finally, this work has sought to establish some direction for future modification of the interface. It is felt that the design of future versions would be best accomplished as

a team effort. The design of a successful interactive environment requires the skills of an artist to identify and define the allusions to things familiar to the user, a person who is skilled at making maximum use of the capabilities of the display hardware, a programmer who can get the most out of the language used to implement the interface, and last but not least, a representative of the target user group to test the ideas generated and to provide input to the implementation effort. The team approach is recommended as it is unusual to find all of these skills in any one individual.

Several problems were encountered during this implementation effort. Their combined effect was to limit the implementation to a skeleton program to serve as a basis for future efforts. These problem areas are discussed next.

B. PROBLEM AREAS

Some of the biggest problems encountered involved becoming familiar with the hardware, specifically the AED 767, and in learning the C programming language and Unix operating system.

One of the early goals for the interface was to allow separate users to each have a directory of design files. It was thought that with the ability to embed Unix systems commands or special purpose "shellscripts" within a program written in C, it would be possible to create and move between several directories of files. A shellscript was written to

accomplish this. However, once control was returned from the shellscript, the environment was returned to the state it was in before the shellscript was called.. Therefore, either the interface program must be called from within the shellscript, or another way to accomplish this goal will have to be found.

Another problem area was with the way the Unix operating system deals with peripheral devices. There are three modes available. They are called "Cooked," "C-Break," and "Raw" in the Unix Manual [Ref. 20]. The default mode is "Cooked" which buffers all input from a terminal until a linefeed character is received. This prevents using single keystrokes for user option selection as each entry must be followed by a "Return" key stroke. Efforts to switch to the "C-Break" or "Raw" modes were unsuccessful but should be pursued in the future.

Several difficulties were encountered while learning to use the capabilities of the AED 767. One of the difficulties was related to the AED's write protect mechanism for the video memory. In theory, this mechanism allows drawing graphics in some color and then protecting those graphics from erasure. This allows a box to be drawn on the screen and then to "protect" the box. Using this mechanism will then permit text to be written in the box and then erased leaving the box on the screen. The documentation implies that all 256 color table locations can be designated as "protected."

However, the video memory is divided into eight memory planes which have the same binary code as color table locations 0, 1, 2, 4, 8, 16, 32, 64, and 128.. Because all other color table location codes correspond to combinations of video memory planes, subtle interactions occur. For example, if you draw a yellow box using the yellow in color table location 3 and then write protect it and afterwards write red text using the red from color table location 1, when you try to erase the text, the erase command will also erase the red component from the yellow box changing it to green. This occurs because the binary codes for 1 and 3 both have the second bit position set to one. The consequence is that a maximum of eight colors can be separately protected at any one time. This is not an insurmountable obstacle, but it does require special attention when write protection is used in the implementation.

There are several difficulties caused by the way the AED's interpreter is addressed. The AED employs a 6502 microprocessor to control the graphics routines. The interpreter mode, which alerts the 6502 to expect a graphics command from the host, is enabled by sending the escape character followed by a sequence of commands. There is a potential problem with this as the escape is a non-printable character and is hard to embed in strings in languages like Pascal. This was not a problem in this implementation as non-printable characters may be included in C language strings.

However, when the interpreter is enabled, the text cursor on the screen is disabled. This requires the implementor to make sure the interpreter gets turned off and on at the appropriate times, and was the source of many irritating implementation delays because forgetting to turn the interpreter on will cause the command string to be printed to the screen but not executed and so on.

The last major problem area involves the complicated nature of CSDL. The variety of options and the recursive nature of the language make it difficult to ensure that all of the capabilities of the language are implemented. The approach used in this implementation was to handle recursion with a loop construct, having the user enter an option to leave the loop, to handle potentially empty parts of the language by giving the user an option of none where appropriate, and to handle the choice of other options by grouping similar types of CSDL constructs together, i.e., the input, output, and duplex specifications of the environment section, and provide separate screen displays to guide the user through entering data for the specific option chosen.

The implementation of an interactive environment is a difficult task which was reaffirmed during the course of this work. There are sure to be problems yet to be encountered and possibly better solutions to those which were encountered as future work is done on this interface.

C. FUTURE WORK

The conclusion was reached during this work that the AED 767 is an excellent graphics device, but is better suited for implementing pictorial design tools such as the "Caesar" VLSI design program. Its graphics routines are highly efficient and the screen can be redrawn very quickly, but higher level interface tools such as windowing and cursor control with a mouse must be created by the implementor. The AED does have a digitizer pad and a joy stick, but these require initialization and control routines to be embedded in any program using them. The Sun Workstation at the Naval Postgraduate School now has built in window routines and a mouse driven cursor. It also runs the Unix operating system. Therefore, it is recommended that future work should be oriented toward taking the foundations laid here and implementing a system to realize them on the Sun Workstation.

In conclusion, the area of creating useful interactive environments in the realm of the human-computer interface is still an area where few absolutes exist. Many opinions can be expressed, and much can be written about the subject in general or any specific interface by itself, but the true test comes with the implementation. In order for the environment to be easy for the human to use, the designer of the environment must carefully consider and hide a multitude of details from the user, and in order for the environment to be a success, the human must want to,and like to, use it.

APPENDIX A

CSDL SYNTAX

```
<letter> ::= A / B / C / D / E / F / G / H / I / J /
             K / L / M / N / O / P / Q / R / S / T /
             U / V / W / X / Y / Z
<digit> ::= 0 / 1 / 2 / 3 / 4 / 5 / 6 / 7 / 8 / 9
<integer> ::= <digit> / <integer><digit>
 real number> ::= <integer> / <integer>. /
                  <integer>.<integer>
<based number> ::= <real number>#<base>
 base> ::= <real number>
<exponential number> ::= <real number>E<real number>
<number> ::= <integer> / <real number> /
             <based number> / <exponential number>
<underscore> ::=
<character> ::= <letter> / <digit> / <underscore>
 id> ::= <letter> / <id><letter> / <id> <digit> /
         <id><Character><id>
<string> ::= "" / <string><letter>"" /
             "" <string><Character>""
<addop> ::= + / -
 mop> ::= * / /
<relational op> ::= < / <= / = / > / >= / /=
<primary> ::= <number> / <string> / <name> /
              ( <expression> )
<factor> ::= <primary> / <factor> ** <primary>
<term> ::= <factor> / <term> <mop> <factor>
 simple exp> ::= <term> / <addop> <term> /
                 <simple exp> <addop> <term>
<relation> ::= <simple exp> /
               <simple exp><relational op><simple exp>
<exp 4> ::= <relation> / <exp 4> AND <relation>
<exp 3> ::= <exp 4> / <exp 3> OR <exp 4>
 exp 2> ::= <exp 3> / <exp 2> => <exp 3>
<expression> ::= <exp 2> / <expression-list> = <exp 2>
<expr list> ::= <> / <expression> /
```

```
                    <expr list> , <expression>
<if then> ::= IF <expression> THEN <stmt el> ENT IF
  while do> ::= WHILE <expression> :
                <max loop count> DO
                <stmt EP> ENT WHILE
 for loop  ::= <for head> DO <stmt EP> END FOR
<for head> ::= FOR <id> FROM <expression>
                TO <expression> : <max loop count>
<perform task> ::= <id> /
                   <id> ( <expr list> : <id list> )
  max loop count> ::= <number>
<left part list> ::= <name> := /
                     <left part list> <name> :=
  assignment stmt> ::= <left part list> <expression>
<data input> ::= SENSE ( <name>
<data output> ::= ISSUE ( <name>
 time measure> ::= H / M / S / MS / US / NS
<period> ::= <number> <time measure>
<time> ::= <period> / <time> <period>
  time block> ::= IN <period> DO stmt EP> END IN
<wait> ::= WAIT <period> /
           WAIT <expression> : <period>
 wait until> ::= WAIT UNTIL <expression> : <period>
<basic stmt> ::= <if then> / <while do> /
                 <for loop> / <perform task> /
                 <data input> / <data output> /
                 <timed block> /
                 <wait> / <wait until>
<labeled stmt> ::= <id> : <basic stmt>
<stmt> ::= <basic stmt> / <labeled stmt>
 stmt EP> ::= <stmt> ; / <stmt EP> <stmt> ;
<proc def> ::= <binary spec> / <allocate spec> /
               <code spec> / <code var spec>
 input spec> ::= INPUT :
                 <name> , <input spec>
```

```
 output spec> ::= OUTPUT :
                  <transmission body> END OUTPUT
<dec> ::= <proc dec> / <input spec> /
          <output spec> / <duplex spec>
<dec gp> ::= <dec> ; / <dec gp> <dec> ;
<duplex spec> ::= DUPLEX :
                  <transmission body> END DUPLEX
<binary spec> ::= BINARY : <binary body> END BINARY
<arithmetic spec> ::= ARITHMETIC :
                      <arithmetic body> END ARITHMETIC
<transmission body> ::= <transmission dec>; /
                        <transmission body>
                        <transmission dec> ;
<transmission dec> ::= <id> ,
                       <binary precision> ,
                       <technology>
<binary body> ::= <binary dec> ; /
                  <binary body> <binary dec> ;
<binary dec> ::= <id>  structure> ,
                 <binary precision> <initial value>
 arithmetic body> ::= <arithmetic dec> ; /
                      <arithmetic body> <arithmetic dec> ;
<arithmetic dec> ::= <id> <structure> ,
                     <decimal precision> <initial value>
<structure> ::= <> /   <number list> )
<number list> ::= <number> / <number list> , <number>
<binary precision> ::= <number>
<decimal precision> ::= <number>
<initial value> ::= <> / , <number>
<technology> ::= RTL / ECL / ITL
<code var spec> ::= CODE VARIABLES :
                    <code dec list> END CODE VARIABLES
 code dec list> ::= <code dec> ;
                    <code dec list> <code dec> ;
<code dec> ::= <id> : <code id>
```

```
<code spec> ::= CODE : <id> , <binary precision> ;
                  <character rep list> END CODE
<character rep list> ::= <character rep> ; /
                         <character rep list> <character rep> ;
<character rep> ::= <id> : <number>
<code id> ::= <id> / ASCII8 / ASCII7 / EBCDIC / BCD
<id list> ::= <> / <id> / <id list> , <id>
<name> ::= <id> / <id> ( <expr list> ) /
           <id> <number> : <number>
<formal param list> ::= <> / ( <id list> : <id list> )
<proc> ::= <task> / <function>
<task> ::= <task head> : <proc dec gp> <stmt gp> END <id>
<proc dec gp> ::= <> / <proc dec> ; /
                  <proc dec gp> <proc dec> ;
<task head> ::= TASK <id> <formal param list>
<function> ::= <function head> ;
               <proc dec gp> <stmt> END <id>
<function head> ::= FUNCTION <id> <formal param list> :
                    BINARY ,
                    <binary precision> <initial value> /
                    FUNCTION <id> <formal param list> :
                    ARITHMETIC ,
                    <decimal precision> <initial value>
<proc gp> ::= <proc> ; / <proc gp> <proc> ;
<proc section> ::= <> / PROCEDURE <proc gp>
<roe> ::= <period>
<t1> ::= <period>
<t2> ::= <period>
<rank> ::= <nz> / <nz> . <pi>
<nz> ::= <number>
<pi> ::= <number>
<qualification> ::= <> / IF <expression>
<episode timing> ::= <> / : <roe> / : <roe> , <t1> /
                     : <roe> , <t1> . <t2> /
                     : <roe> , <t1> , <t2> , <roe>
```

```
<when do> ::= <qualification> WHEN
              <name> <episode timing> DO <task list>
<simple do> ::= <qualification> DO <task list> <name>
<every> ::= <qualification> EVERY <roe> DO <task list>
<at time> ::= <qualification> AT <time> DO <task list>
<task list> ::= <name> / <task list> THEN <name>
<contingency def> ::= <when do> / <simple do> /
                      <every> / <at time>
<list body> ::= <contingency def> ; /
                <list body> <contingency def> ;
<contingency list> ::= <> / CONTINGENCY LIST <list body>
<design criteria> ::= <> / DESIGN CRITERIA METRIC <metric> ;
                      VOLUMES <number list> ;
                      MONITORS <number list> ;
<metric> ::= FIRST / COST / POWER
<environment section> ::= <> / ENVIRONMENT <dec ch>
<id section> ::= <> / IDENTIFICATION
                 DESIGNER : <string> DATE : <string>
                 PROJECT : <string>
<control system design> ::= <id section> <design criteria>
                            <environment section>
                            <proc section>
                            <contingency list>
 system goal symbol> ::= <control system design> END
```

APPENDIX B

IMPLEMENTATION PROGRAM

```
/* This is the main program for the CSDL interface.
   This program calls several functions which have
   been divided into CSDL design functions and
   graphics utility functions.                        */

#include <stdio.h>

#define MAXLINE 1024

main()
  {
    char c;

    FILE *fopen(), *fl;
    fl = fopen("output.csde", "w");

    initialize_ped();
    boundry();
    welcome();
    prmtl1();

/* This block of code waits for the user to make a choice from
   those offered in the user prompt. Based on the choice made
   the user will be presented a new screen.                   */

    while ((c = getchar()) != 'q')
    {
      switch(c)
      {
        case 's':
```

```
workarea();
heading("SYSTEM INITIALIZATION");
 printf("\n");
 procmiz();
 break;
 case 'i':
 workarea();
 in_section(f1);
 procmit();
 break;
 case 'd':
 workarea();
 criteria_section(f1);
 promit();
 break;
 case 'e':
 workarea();
 environment_section(f1);
 promit();
 break;
 case 'c':
 workarea();
 contingency_section(f1);
  promit();
 break;
 case 'p':
 workarea();
 procedures_section f1,;
 promit();
 break;
 default:
 break;
 }
 printf("\n");
```

```
printf("FFD");
printf("");
printf("SECØ.");
find_line(23);
printf("XXXThe colored boxes show the Ø);
printf("current colors that are defined in Ø);
printf("the color table. The color table locations Ø);
printf("shown are 1 through 8, 16, 32 through 35, Ø);
printf("64, and 128. These locations correspond to Ø);
printf("the boxes from left to right. Color table Ø);
printf("location Ø is the background color. Ø);
printf("The colors are set in the function Ø);
printf("initialize_aed and may be changed there. Ø);
printf("");
printf("MOV5Ø.5Ø.SEC01.DFR75.75.");
printf("MOV8Ø.5Ø.SEC02.DFR1Ø5.75.");
printf("MOV11Ø.5Ø.SEC03.DFR135.75.");
printf("MOV14Ø.5Ø.SEC04.DFR165.75.");
printf("MOV17Ø.5Ø.SEC05.DFR195.75.");
printf("MOV2ØØ.5Ø.SEC06.DFR225.75.");
printf("MOV23Ø.5Ø.SEC07.DFR255.75.");
printf("MOV26Ø.5Ø.SEC08.DFR285.75.");
printf("MOV29Ø.5Ø.SEC16.DFR315.75.");
printf("MOV32Ø.5Ø.SEC32.DFR345.75.");
printf("MOV35Ø.5Ø.SEC33.DFR375.75.");
printf("MOV38Ø.5Ø.SEC34.DFR4Ø5.75.");
printf("MOV41Ø.5Ø.SEC35.DFR435.75.");
printf("MOV44Ø.5Ø.SEC64.DFR465.75.");
printf("MOV47Ø.5Ø.SEC128.DFR495.75.");
printf("SECØ.");
find_line(22);
close(f1);
printf("XXXThe CSIL text for the design Ø);
printf("is in the file named output.csil. Ø);
printf("");
```

```
    printf "MOV#.#.XXX" ;
  }


/* This module initializes the AED 767 to receive
   graphics commands from the host computer.   */

initialize_aed()
  {
   printf("");
   printf("SEN3DDDN");
   printf "FFD";
   printf("");
   printf("SCT#.3.2.2.67.255.150.177.0.167.0.");
   printf "SCT3.1.244.75.102.");
   printf("SCT4.1.255.255.255.");
   printf("SCI16.1.200.250.102.");
   printf "SCT32.1.75.104.204.");
   printf("SCT64.1.150.167.200.");
   printf("SCT128.1.3.3.255.");
   printf "SCT3.3.128.128.128.0.104.128.150.100.255.");
    /* include DSK and here when functions are chosen */
   return;
  }

/* This next block of code sets
   the text size and the text color.
   It then fprintfs a title on the screen.  */

welcome()
  {
   printf "";
   printf("SEC8.SAP270.77.L");
   printf("MOV356,514.");
   printf "XXATHP";
```

```
    printf("");
    printf("MOV194.450.");
    printf("XXXCOMPUTER SYSTEM DESIGN");
    printf("");
    printf("MOV274.390.");
    printf("XXXENVIRONMENT");
    printf("");
    printf("MOV344.330.");
    printf("XXXCSDL");
    printf("");
    printf("SAP178.13.L");
    printf("MOV302.270.");
    printf("XXXCOMPUTER AIDED DESIGN");
    printf("");
    printf("MOV370.240.");
    printf("XXXOF");
    printf("");
    printf("MOV322.210.");
    printf("XXXMICROPROCESSOR-BASED");
    printf("");
    printf("MOV226.180.");
    printf("XXXCONTROL SYSTEMS");
    return;
}

/* This section contains the functions which
   control the input of a design in CSDL.    */

/*##########################################################
### These functions handle the Identification      ###
### Section of a design in CSDL.                   ###
##########################################################*/


id text,
```

```
{
 printf("");
 printf "SEC13.");
 printf("MAR6.552.");
 find line(5);
 printf "XXXThe purpose of the Identification`);
 printf(" Section is to record the ");
 printf "designer's name, the date, and the");
 printf(" title of the design project. ");
 printf("Please enter the appropriate information");
 printf "as it is requested.");
 printf("");
 printf("MAR0.0.");
 return;
}


id_section(f1)
 {
  int i;
  char c;
  char name[50], date[15], project[60];

  c = getchar();
  heading("IDENTIFICATION SECTION");
  id text 2;
  printf("");
  fprintf(f1," IDENTIFICATION ");
  printf("");
  find line(12);
  printf("XXANAME: ");
  fprintf f1,"    DESIGNER : ");
  getline(name);
  fprintf(f1,"%s ",name);
  printf("");
```

```
find_line(15);
fprintf(f1,"XXDATE: ");
getline(date);
fprintf(f1,"    DATE : ");
fprintf(f1,"%s %,date);
fprintf(f1,"\n");
find_line(15);
fprintf(f1,"XXPROJECT: ");
getline(project);
fprintf(f1,"    PROJECT : ");
fprintf(f1,"%s %,project);
fprintf(f1,"\n");
  return;
}

/*##########################################################
###  End Identification Section                        ###
##########################################################*/


/*##########################################################
### These functions handle the design Criteria         ###
### Section of a design in CSDL.                       ###
##########################################################*/

criteria_text()
{
  fprintf("\n");
  fprintf("SCSIG.\n");
  fprintf("MARC.552.\n");
  find_line(5);
  fprintf("XXTHE purpose of the design criteria");
  fprintf(" Section is to allow the ");
  fprintf(" designer to specify the criteria");
```

```
    printf " for his design. This includes Ø);
    printf("a criteria metric, a list of volumes");
    printf " and a list of monitors Ø);
    printf("for DSDB to consider. Ø ;
    printf("Please make the desired choice from the");
    printf " list at the right." ;
    printf("");
    printf("^ARC.Ø.");
    return:
  }


criteria_section(f1)
 {

  char a, b, c;
  int i, j, u, v, w, x;
  int volumes, monitors;

  volumes = 3;
  monitors = 3;

  c = getchar();
  heading("DESIGN CRITERIA SECTION");
  criteria_text();
  printf("");
  metric_menu();
  printf("SEC16.");
  fprintf(f1," DESIGN CRITERIA Ø,;
  printf("");
  fprintf(f1,"    METRIC ");
  printf("");
  scanf("%d",&u);
  while(u < 1 || u > 3)
   {
```

```
  if((a < 1) || (a > 3))
   {
    printf("");
    printf("SEC1.");
    find_line(24);
    printf("XXXPlease choose one from the");
    printf("given list.");
    printf("");
    printf("SEC0.");
    find_line(23);
    printf("XXXMake choice: %d",v);
    printf("");
    printf("SEC1.");
    find_line(23);
    printf("XXXMake choice: ");
    scanf("%d",&u);
    printf("SEC15.");
   }
 }
switch (u)
 {
  case 1:
   fprintf(f1,"FIRST ; 0);
   break;
  case 2:
   fprintf(f1,"COST ; 0);
   break;
  case 3:
   fprintf(f1,"POWER ; 0);
   break;
  default:
   break;
 }
erase_menu();
volumes_menu();
```

```
fprintf(f1,"    VOLUMES ");
printf("");
printf("S=C1.");
find_line(23);
printf("XXXMake first choice: ");
scanf("%d",&v);
printf("");
if(v == 0)
 fprintf(f1,"1, 2, 3 ; 0,");
else
 {
  for(i = 1; i <= volumes; i++)
   {
    while((v < 0) || (v > volumes))
     {
      if (v < 1) || (v > volumes)
       {
        printf("");
        printf("S=C1.");
        find_line(23);
        printf("XXXPlease choose one from the");
        printf("given list.");
        printf("");
        printf("S=C1.");
        find_line(23);
        printf("XXXMake first choice: %d",v);
        printf("");
        printf("S=C1.");
        find_line(23);
        printf("XXXMake first choice: ");
        scanf("%d",&v);
        printf("");
       }
     }
    if (v ==
```

```
  {
   fprintf(f1,"1, 2, 3 ; \n");
   break; /* break out of for loop */
  }
printf("");
printf("SUB1.");
find_line(25 + i);
if(i < volumes)
  {
   printf("XXXMake choice # %d : ",i+1);
  }
switch (i)
  {
   case 1:
    switch (v)
     {
      case 1:
       fprintf(f1,"1, ");
       break;
      case 2:
       fprintf(f1,"2, ");
       break;
      case 3:
       fprintf(f1,"3, ");
       break;
      default:
       break;
     }
    break;
   case 2:
    switch (v)
     {
      case 1:
       fprintf(f1,"1, ");
       break;
```

```
case 2:
  fprintf(f1,"2, ");
  break;
case 3:
  fprintf(f1,"3,");
  break;
default:
  break;
}
case 1:
  fprintf(f1,"1 ");
  break;
case 2:
  fprintf(f1,"2 ");
  break;
case 3:
  fprintf(f1,"3 ");
  break;
default:
  break;
}
```

```
fprintf(f1,"    MONITORS ");
printf("");
printf("SEC1.");
find line(25);
printf("XXXMake first choice: ");
scanf("%d",&w);
printf("");
if(w == 0)
 fprintf(f1,"1, 2, 3 ; 0");
else
 {
  for(j = 1; j <= monitors; j++,
   {
    while( w < 0) || (w > monitors,)
     {
      if((w < 1) || (w > monitors))
       {
        printf("");
        printf("SEC1.");
        find line(20);
        printf("XAXPlease choose one from the");
        printf(" given list.");
        printf("");
        printf("SEC2.");
        find line(25);
        printf("XXXMake first choice: %d",w);
        printf("");
        printf("SEC1.");
        find line(25);
        printf("XAXMake first choice: ");
        scanf("%d",&w);
        printf("");
       }
     }
   }
```

```
  {
   fprintf(f1,"1, %d, %d ; ",i,j);
   break; /* Break out of for loop */
  }
 printf("");
 printf("SIC1.");
 find_line(25 + j);
 if (j < monitors)
  {
   printf("\n\nMake choice # %d : ",j+1);
  }
 switch (j)
  {
   case 1:
    switch (w)
     {
      case 1:
       fprintf(f1,"1, ");
       break;
      case 2:
       fprintf(f1,"2, ");
       break;
      case 3:
       fprintf(f1,"3, ");
       break;
      default:
       break;
     }
    break;
   case 2:
    switch (w)
     {
      case 1:
       fprintf(f1,"1, ");
       break;
```

```
        case 2:
          fprintf(f1,"2 ");
          break;
        case 3:
          fprintf(f1,"3 ");
          break;
        default:
          break;
      }
    break;
  case 3:
    switch (n)
    {
      case 1:
        fprintf(f1,"1 ; ");
        break;
      case 2:
        fprintf(f1,"2 ; ");
        break;
      case 3:
        fprintf(f1,"3 ; ");
        break;
      default:
        break;
    }
    break;
  default:
    break;
}
if(n < 3)
  scanf("%d",&n);
}
fprintf(f1,"\n");
fclose(f1);
```

```
    return;
}


metric_menu()
{
    printf("");
    printf("SEC1." );
    menu_lines(1);
    printf("MAP565.757.");
    printf("XXXThe metrics currently a");
    printf("available are: ");
    printf("");
    menu_lines(5);
    printf("XXX1. FIRST");
    printf("");
    menu_lines(7);
    printf("XXX2. COST");
    printf("");
    menu_lines(9);
    printf("XXX3. POWER");
    printf("");
    menu_lines(11);
    printf("XXXMake a choice by typing a");
    printf("the number by the a");
    printf("desired metric.");
    printf("");
    printf("MAP2.0.");
    return;
}


volumes_menu()
{
    printf("");
```

```
    printf "SEC1.";
    menu_lines(1);
    printf("MAB55.757.");
    printf "XXXThe volumes currently &);
    printf("available are: &);
    printf("");
    menu_lines(5);
    printf("XXXC. Default Order (1,2,3)",;
    printf("");
    menu_lines(7);
    printf("XXX1. First");
    printf("");
    menu_lines(9);
    printf("XXX2. Second");
    printf("");
    menu_lines(11);
    printf("XXX3. Third");
    printf("");
    menu_lines(13);
    printf("XXXEnter the numbers of &);
    printf "the volumes in the &.:
    printf("order they are to be &);
    printf("considered. &);
    printf("");
    printf("MAR&.3.");
    return;
  }


monitors menu()
  {
    printf("");
    printf "SEC1.";
    menu_lines(1);
```

```
printf("\nPress.ESC.");
printf("\n\nThe monitors currently 2));
printf("available are: 2));
printf("\n");
menu_lines(1);
printf("\n\n. Default Order '1,2,3)");
printf("\n");
menu_lines(2);
printf("\n1. First");
printf("\n");
menu_lines(3);
printf("\n2. Second");
printf("\n");
menu_lines(11);
printf("\n3. Third");
printf("\n");
menu_lines(15);
printf("\nEnter the numbers of
printf("the monitors in the
printf("order they are to be
printf(" considered.");
printf("\n");
printf("\n\n");
return;
}
```

```
environment_text()
 {
  printf("");
  printf "SFC16.");
  printf("MARG.F!2.");
  find line(5);
  printf("XXXThe purpose of the Environment");
  printf(" Section is to allow the d);
  printf "designer to specify the input and");
  printf(" output signals, arithmetic and Ø);
  printf("binary constants, and code symbols");
  printf " for his design. The environment 2);
  printf("section may be left empty, or may ");
  printf(" have one or more of the types of 2);
  printf "parameters shown. 2);
  printf("Please make the desired choices from the");
  printf(" list at the right.");
  printf("");
  printf("MARG.?.");
  return;
 }


environment_menu1()
 {
  printf("");
  printf "SFC1.");
  menu_lines(1);
  printf("MARGGG.757.");
  printf "YY(The environment is Y");
  printf("composed of the Y);
  printf "following: .");
  printf("");
  menu_lines(5);
  printf "XXXZ. EMPTY");
```

```
  printf("");
  menu_lines(8);
  printf("XXX1. INPUT and OUTPUT Ø");
  printf("     PARAMETERS");
  printf("");
  menu_lines(11);
  printf("XXX2. CONSTANTS");
  printf("");
  menu_lines(14);
  printf("XXX3. COLORS");
  printf("");
  menu_lines(17);
  printf("XXX4. FINISHED");
  printf("");
  menu_lines(22);
  printf("XXXMake a choice by typing Ø");
  printf("the number by the Ø");
  printf("desired category.");
  printf("");
  printf("MARK.Ø.");
  find_line(25);
  printf("XXXChoice: ");
  return;
 }


environment_menu()
 {
  printf("");
  printf("SΣC1.");
  menu_lines(1);
  printf("MAIN.7:7.");
  printf("XXXThe following options Ø");
  printf("are now available: Ø");
  printf("");
```

```
    menu_lines(5);
    printf("XXX1. INPUT and OUTPUT ");
    printf("   PARAMETERS");
    printf("");
    menu_lines(8);
    printf("XXX2. CONSTANTS");
    printf("");
    menu_lines(11);
    printf("XXX3. CODES");
    printf("");
    menu_lines(14);
    printf("XXX4. FINISHED");
    printf("");
    menu_lines(17);
    printf("XXXMake a choice by typing ");
    printf("the number by the ");
    printf("desired category.");
    printf("");
    printf("MAP0.0.");
    find_line(23);
    printf("XXXChoice: ");
    return;
  }


environment_section(f1)

 int *f1;

 {

  char c;
  int c, done;

  c = getchar();
    .
    .
```

```
heading("ENVIRONMENT SECTION");
environment_text();
environment_menu1();
scanf("%d",&u);
printf("");
if(u == 2)
 {
  erase_menu();
  prompt2();
 }
else
 {
  done = 2;
  fprintf(f1," ENVIRONMENT 2");
  printf("");
  while(done == 2)
   {
    switch (u)
     {
      case 1:
       signals(f1);
       heading("ENVIRONMENT SECTION" ;
       environment_text();
       environment_menu2();
       done = 2;
       scanf("%d",&u);
       printf("");
       break;
      case 2:
       constants(f1);
       heading("ENVIRONMENT SECTION" ;
       environment_text();
       environment_menu2();
       done = 2;
       scanf("%d",&u);
```

```
printf("");
break;
case 3:
codes(fl);
heading("ENVIRONMENT SECTION");
environment_text();
environment_menu();
done = 4;
scanf("%d",&c);
printf("");
break;
case 4:
done = 1;
break;
default:
while((u < 1 || (u > 4)
{
  printf("");
  printf("SEC1.");
  find_line(2);
  printf("XXXPlease choose one from the");
  printf("...given list.");
  printf("");
  printf("SEC2.");
  find_line(2);
  printf("XXXchoice: ");
  printf("");
  printf("SEC1.");
  find_line(2);
  printf("XXXchoice: ");
  scanf("%d",&u);
  printf("");
}
printf("SEC2.");
find_line(2);
```

```
            printf("XXXPlease choose one from the");
            printf("given list.");
            printf("");
            printf("SEC1.");
            done = 0;
            break;
          }
        }
      }
    fprintf(f1,"");
    erase_menu();
    rom_pt3();
    return;
  }


signals_text()
  {
    printf("");
    printf("SEC15.");
    printf("MAKS.Box.");
    find line(5);
    printf("XXXThe signals used to communicate");
    printf(" with the controller can be ");
    printf("classified as being INPUT, OUTPUT,");
    printf(" or DUPLEX. Each signal is identified ");
    printf("by an id name, a binary precision,");
    printf(" and a technology descriptor. ");
    printf("Section may be left empty, or may");
    printf(" have one or more of the types of ");
    printf("signals shown. ");
    printf("Please make the desired choices from the");
    printf("list at the right.");
    printf("");
```

```
    printf "MARK.0.");
    return;
  }

signals_menu1()
  {
   printf("");
   printf("SEC1.");
   menu_lines 1);
   printf("MAR565.757.");
   printf("XXXSignals may be 2);
   printf "classified as one of 4);
   printf("the following: 0);
   printf "");
   menu_lines(5);
   printf("XXX3. EMPTY");
   printf "");
   menu_lines(8);
   printf("XXX1. INPUT 2");
   printf "");
   menu_lines(11);
   printf("XXX2. OUTPUT");
   printf "");
   menu_lines(14);
   printf("XXX3. DUPLEX");
   printf "");
   menu_lines(17);
   printf("XXX4. FINISHED");
   printf "");
   menu_lines(20);
   printf "XXXMake a choice by typing 0);
   printf "the number by the 4);
   printf("desired category.");
   printf "");
   printf "MARK.0.";
```

```
    find_line(25);
    printf("XXXChoice: ");
    return;
   }


signals_menu2()
 {
   printf("");
   printf("SEC1.");
   menu_lines(1);
   printf("MAR565.757.");
   printf("XXXValid choices are 2");
   printf("now the following: 0");
   printf("");
   menu_lines(5);
   printf("XXX1. INPUT 2");
   printf("");
   menu_lines(8);
   printf("XXX2. OUTPUT");
   printf("");
   menu_lines(11);
   printf("XXX3. BUFFY");
   printf("");
   menu_lines(14);
   printf("XXX4. FINISHED");
   printf("");
   menu_lines(17);
   printf("XXXMake a choice by typing ");
   printf("the number by the ");
   printf("desired category.");
   printf("");
   printf("MAR...");
   find_line(25);
   printf("XXXChoice: ");
```

```
  return;
 }

signals(f1)

 int *f1;

 {

  int v, d2;

  erase_text();
  heading("INPUT/OUTPUT PARAMETERS");
  signals_text();
  signals_menu();
  scanf("%d",&v);
  printf("");
  if(v == 0)
   {
    erase_text();
   }
  else
   {
    d2 = 0;
    printf("");
    while(d2 == 0)
     {
      switch (v)
       {
        case 1:
         input_parameters(f1);
         heading("INPUT/OUTPUT PARAMETERS");
         signals_text();
         signals_menu();
```

```
 d2 = 0;
 scanf("%d",&v);
 printf("");
 break;
case 2:
 output_parameters(f1);
 heading("INPUT/OUTPUT PARAMETERS");
 signals_text();
 signals_menu2();
 d2 = 0;
 scanf("%d",&v);
 printf("");
 break;
case 3:
 duplex_parameters(f1);
 heading("INPUT/OUTPUT PARAMETERS");
 signals_text();
 signals_menu2();
 d2 = 0;
 scanf("%d",&v);
 printf("");
 break;
case 4:
 d2 = 1;
 break;
default:
 while((v < 1) || (v > 4))
  {
   printf("");
   printf("ERR1.");
   find_line(24);
   printf("\nPlease choose one from the");
   printf(" given list.");
   printf("");
   printf("ERR.");
```

```
                find_line(25);
                printf("XXXChoice: %d",v);
                printf("");
                printf("SEC1.");
                find_line(25);
                printf("XXXChoice: ");
                scanf("%d",&v);
                printf("");
              }
             d2 = 4;
             break;
           }
        }
      }
    printf(f1,"3");
    erase_text();
    return;
  }


input_text()
  {
   printf("");
   printf "SEC1C.");
   printf("MARG.552.");
   find line(5);
   printf "XXXEach input signal is to be");
   printf(" specified by an identifier,");
   printf "number indicating the binary");
   printf(" precision of the signal, and");
   printf("the type of technology used to");
   printf "generate that signal. ");
   printf("The identifiers used to name the");
   printf(" signals must begin with a letter");
   printf "and should be no longer than");
```

```
    printf(" eight characters. \n");
    printf("Please enter the appropriate information");
    printf(" as it is requested. \n");
    printf("Please follow the instructions listed");
    printf(" to the right. \n");
    printf("");
    printf("MARK.2.");
    return;
  }


input_menu1()
  {
    printf("");
    printf("SEC1.");
    menu_lines(1);
    printf("MARSOF.757.");
    printf("XXXInput signal \n");
    printf("parameters may be \n");
    printf("entered using the \n");
    printf("following: \n");
    printf("");
    menu_lines(7);
    printf("XXXX. NONE");
    printf("");
    menu_lines(12);
    printf("XXX1. ENTER INPUT \n");
    printf("    PARAMETERS");
    printf("");
    menu_lines(14);
    printf("XXX2. FINISHED");
    printf("");
    menu_lines(16);
    printf("XXXMAKE A CHOICE BY TYPING \n");
    printf("THE NUMBER TO THE ");
```

```
printf("Desired category.");
printf("");
printf("MARGIN.");
find_line(25);
printf("XXXChoice: ");
return;
}

init_menu2()
{
 printf("");
 printf("SEC1.");
 menu_lines(1);
 printf("MARGIN.757.");
 printf("XXXValid choices ");
 printf("now are: ");
 printf("");
 menu_lines(7);
 printf("XXX1. ENTER INPUT ");
 printf("   PARAMETERS");
 printf("");
 printf("");
 menu_lines(12);
 printf("XXX2. FINISHED");
 printf("");
 menu_lines(14);
 printf("XXXMake a choice by typing ");
 printf("the number by the ");
 printf("desired category.");
 printf("");
 printf("MARGIN.");
 find_line(23);
 printf("XXXChoice: ");
 return;
}
```

```
input_parameters f1)

 int *f1;

 {

  char c, id[23], precision[10];
  int w, x, y, d2, d4;

  erase_text();
  heading("INPUT SIGNAL PARAMETERS");
  input_text();
  input_menu1();
  scanf("%d",&w);
  printf("");
  if w == 0)
   {
    erase_text();
   }
  else
   {
    d3 = 0;
    fprintf(f1,"    INPUT : 0.");
    printf("");
    while(d3 == 0)
     {
      switch (w)
       {
        case 1:
         printf("");
         find_line(15);
         printf("XXXSIGNAL Id:  ");
         c = getchar();
         getline(id);
         fprintf(f1,"      ",id);
```

```
printf("");
find_line(17 ;
printf("XXXPRIMARY PRECISION: ");
scanf("%d",&y);
fprintf f1, "%d , ",y);
printf("");
technology_menu();
printf("");
printf("SEC0.");
find_line(25,;
printf("XXXChoice: %d",w);
printf("");
printf("SEC1.");
find_line(21);
printf("XXXChoose from the list ");
printf("at the right.");
printf("");
find_line(24);
printf("XXXTECHNOLOGY: " ;
scanf("%d",&x);
d4 = 0;
while(d4 == 0)
 {
  switch (x)
   {
    case 1:
     fprintf(f1," TTL ;");
     d4 = 1;
     break;
    case 2:
     fprintf(f1," ECL ;" ;
     d4 = 1;
     break;
    case 3:
     fprintf(f1," I2L ;" ;
```

```
        d4 = 1;
        break;
      default:
        while((x < 1) || (x > 3))
          {
            printf("");
            printf("SEC1.");
            find_line(24);
            printf("XXXPlease choose one from the");
            printf("given list.");
            printf("");
            printf("SEC2.");
            find_line(22);
            printf("XXX                    d",x);
            scanf("%d",&x);
            printf("");
            printf("SEC7.");
            find_line(22);
            printf("XXXPlease choose one from the");
            printf("given list.");
            printf("");
            printf("SEC1.");
          }
        d4 = 2;
        break;
    }
  }
fprintf(f1,"");
erase_text();
heading("INPUT SIGNAL PARAMETERS");
input_text();
input_menu();
d3 = 3;
scanf("%d",&w);
printf("");
```

```
     break;
    case 2:
     fprintf(f1,"    END INPUT ");
     d3 = 1;
     break;
    default:
     while((w < 1) || (w > 2))
      {
       printf("");
       printf("S#C1.");
       find_line(22);
       printf("XXXPlease choose one from the");
       printf("given list.");
       printf("");
       printf("S#C2.");
       find_line(25);
       printf("XXXChoice: %d",w);
       printf("");
       printf("S#C1.");
       find_line(25);
       printf("XXXChoice: ");
       scanf("%d",&w);
       printf("");
       printf("S#C4.");
       find_line(23);
       printf("XXXPlease choose one from the");
       printf("given list.");
       printf("");
       printf("S#C1.");
      }
     d3 = 4;
     break;
   }
  }
 }
```

```
    fprintf(fl,"Z);
    erase_text();
    return;
  }


output_text()
  {
   printf("");
   printf("SMC16.");
   printf("MAR9.B52.");
   find_line(8);
   printf("XXXEach output signal is to be");
   printf(" specified by an identifier, Z) ;
   printf("number indicating the binary,");
   printf(" precision of the signal, and Z);
   printf("the type of technology used to");
   printf(" generate that signal. Z");
   printf("The identifiers used to name the");
   printf(" signals must begin with a letter .);
   printf("and should be no longer than");
   printf(" eight characters. Z);
   printf("Please enter the appropriate information");
   printf(" as it is requested. Z);
   printf("Please follow the instructions listed");
   printf(" to the right. Z);
   printf("");
   printf("MAR7.Y.");
   return;
  }


output_menu()
  {
```

```
    printf("");
    printf("SEC1.");
    menu_lines(1);
    printf("VAR165.757.");
    printf("XXXOutput signal ");
    printf("parameters may be ");
    printf("entered using the ");
    printf("following: ");
    printf("");
    menu_lines(7);
    printf("XXX0. NONE");
    printf("");
    menu_lines(10);
    printf("XXX1. ENTER OUTPUT ");
    printf("   PARAMETERS");
    printf("");
    menu_lines(14);
    printf("XXX2. FINISHED");
    printf("");
    menu_lines(18);
    printf("XXXMake a choice by typing ");
    printf("the number by the ");
    printf("desired category.");
    printf("");
    printf("VAR2.0.");
    find_line(25);
    printf("XXXChoice: ");
    return;
}

output_menu2()
{
    printf("");
    printf("SEC1.");
```

```
    menu_lines(1);
    printf("MAR555.757.");
    printf("XXXValid choices 2);
    printf("now are: 2);
    printf("");
    menu_lines(7);
    printf("XXY1. ENTER OUTPUT 0);
    printf("    PARAMETERS");
    printf("");
    menu_lines(12);
    printf("XXY2. FINISHED");
    printf("");
    menu_lines(14);
    printf("XXXMake a choice by typing 2);
    printf("the number by the 2);
    printf("desired category.");
    printf("");
    printf("MAR0.2.");
    find_line(25);
    printf("XX(Choice: ");
    return;
}

output_parameters(f1)

int *f1;

{

    char c, id[22], precision[12];
    int w, x, y, d2, e4;

    erase_text();
    reading("OUTPUT SIGNAL PARAMETERS");
```

```
output text();
output menu1();
scanf("%d",&w);
printf("");
if(w == 0)
 {
  erase text();
 }
else
 {
  d3 = 2;
  fprintf(f1,"    OUTPUT : ");
  printf("");
  while(d3 == 0)
   {
    switch (w)
     {
      case 1:
       printf("");
       find line(15);
       printf("XXXSIGNAL ID: ");
       c = getchar();
       getline(id);
       fprintf(f1,"      %s , ",id);
       printf("");
       find line(17);
       printf("XXXBINARY PRECISION: ");
       scanf("%d",&y);
       fprintf(f1,"%d , ",y);
       printf("");
       technology menu();
       printf("");
       printf("SECO.");
       find line(25 :
       printf("XXXChoice: %d",w);
```

```
printf("");
printf("SEC1.");
find_line(21);
printf("XXXChoose from the list ");
printf("at the right.");
printf("");
find_line(20);
printf("XXXTECHNOLOGY: ");
scanf("%d",&x);
d4 = 0;
while(d4 == 0)
 {
  switch (x)
   {
    case 1:
     fprintf(f1," TTL ;");
     d4 = 1;
     break;
    case 2:
     fprintf(f1," ECL ;" ;
     d4 = 1;
     break;
    case 3:
     fprintf(f1," IIL ;");
     d4 = 1;
     break;
    default:
     while((x < 1) || (x > 3)
      {
       printf("");
       printf("SEC1.");
       find_line(24);
       printf("XXXPlease choose one from the");
       printf(" given list.");
       printf("");
```

```
        printf("SEC2.");
        find_line(22);
         printf("XXX                    ed",x);
        printf("");
        printf("SEC1.");
        scanf("%d",&x);
         printf("");
          }
        d4 = 2;
          break;
            }
      }
    fprintf(f1,"");
    erase text();
    reading("OUTPUT SIGNAL PARAMETERS");
    output_text();
    output_menu2();
    d2 = 0;
    scanf("%d",&w);
    printf("");
    break;
    case 2:
    fprintf(f1,"     END OUTPUT ");
    d3 = 1;
    break;
    default:
    while((w < 1) || (w > 2),
      {
        printf("");
        printf("SEC1.");
        find_line(22);
        printf("XXXPlease choose one from the" );
        printf("given list.");
        printf("");
        printf("SEC2.");
```

```
                find_line(25);
                printf("XXXChoice: %d",w);
                printf("");
                printf("SEC1.");
                find_line(25);
                printf("XXYChoice: ");
                scanf("%d",&w);
                printf("");
              }
            d2 = 0;
            break;
          }
        }
      }
    fprintf(fi,"A");
    erase_text();
    return;
  }

duplex_text()
  {
  printf("");
  printf("SEC16.");
  printf("MAR9.552.");
  find_line(5);
  printf("XXYEach duplex signal is to be ");
  printf("specified by an identifier, ");
  printf("number indicating the binary ");
  printf("precision of the signal, and ");
  printf("the type of technology used to ");
  printf("generate that signal. ");
  printf("The identifiers used to name the");
  printf(" signals must begin with a letter ");
  printf("and should be no longer than ");
```

```
printf(" eight characters. \n");
printf("Please enter the appropriate information");
printf(" as it is requested. \n");
printf("Please follow the instructions listed");
printf("to the right. \n");
printf("");
printf("MARK.");
return;
}


duplex_menu1()
{
printf("");
printf("SEC1.");
menu_lines(1);
printf("MAR565.757.");
printf("XXXDuplex signal \n");
printf("parameters may be \n");
printf("entered using the \n");
printf("following: \n");
printf("");
menu_lines(7);
printf("XXX2. NONE");
printf("");
menu_lines(10);
printf("XXX1. ENTER AN DUPLEX \n");
printf("   PARAMETER");
printf("");
menu_lines(14);
printf("XXX2. FINISHED");
printf("");
menu_lines(18);
printf("XXXMake a choice by typing \n");
printf("the number by the ");
```

```
    printf("desired category.");
    printf("");
    printf("MAR8.2.");
    find_line(25);
    printf("XXXChoice: ");
    return;
  }


duplex_menu2()
  {
    printf("");
    printf("SEC1.");
    menu_lines(1);
    printf("MAR555.757.");
    printf("XXXValid choices M");
    printf("now are: M");
    printf("");
    menu_lines(7);
    printf("XXX1. ENTER AN DUPLEX M");
    printf("    PARAMETER");
    printf("");
    menu_lines(10);
    printf("XXX2. FINISHED");
    printf("");
    menu_lines(14);
    printf("XXXMake a choice by typing M");
    printf("the number by the M");
    printf("desired category.");
    printf("");
    printf("MAR2.2.");
    find_line(25);
    printf("XXXChoice: ");
    return;
  }
```

```
duplex_parameters f1)

 int *f1;

 {

  char c, id[4x];
  int w, x, y, d3, d4;

  erase_text();
  heading("DUPLEX SIGNAL PARAMETERS");
  duplex_text();
  duplex_menu1();
  scanf("%d",&w);
  printf("");
  if(w == 0)
   {
    erase_text();
   }
  else
   {
    d3 = 0;
    fprintf(f1,"    DUPLEX : 0");
    printf("");
    while(d3 == 0)
     {
      switch (w)
       {
        case 1:
         printf("");
         find_line(15);
         printf("XXXSIGNAL ID: ");
         c = getchar();
         getline(id);
         fprintf f1,"      %c ,  ",id ;
```

```
printf("");
find_line(17,;
printf("XXXBINARY PRECISION: ");
scanf("%d",&y);
fprintf(f1," %d , ",y);
printf("");
technology_menu();
printf("");
printf("SEC0.");
find_line(22,;
printf("XXYChoice: %d".x);
printf("");
printf("SEC1.");
find_line(21);
printf("XXXChoose from the list ");
printf("at the right.");
printf("");
find_line(20);
printf("XXYTECHNOLOGY: ",;
scanf("%d",&x);
d4 = 2;
while(d4 == 2)
 {
  switch (x)
   {
    case 1:
     fprintf(f1," TTL ;");
     d4 = 1;
     break;
    case 2:
     fprintf f1," ECL ;");
     d4 = 1:
     break;
    case 3:
     fprintf f1," ITL ;" ;
```

```
        d4 = 1;
        break;
      default:
        while((x < 1) || (x > 3))
          {
            printf("");
            printf("SEC1.");
            find_line(24);
            printf("XXXrlease choose one from the");
            printf("     given list.");
            printf("");
            printf("SEC2.");
            find_line(24);
            printf("XXX                    xa",x);
            printf("");
            printf("SEC1.");
            scanf("%d",&x);
            printf("");
          }
        d4 = 2;
        break;
      }
    }
  fprintf(f1," 2");
  erase_text();
  reading("DUPLEX SIGNAL PARAMETERS");
  duplex_text();
  duplex_menu2();
  d2 = 2;
  scanf("%d",&x);
    clrtf("");
  break;
  case 2:
  fprintf(f1,"                    END DUPLEX 2");
  d2 = 1;
```

```
        break;
    default:
        while((w < 1) || (w > 4))
            {
              printf("");
              printf("SEC1.");
              find_line(22);
              printf("XXXPlease choose one from the");
              printf(" given list.");
               printf("");
              printf("SEC2.");
              find_line(23);
              printf("XXXChoice: %d",w);
              printf("");
              printf("SEC1.");
              find_line(25);
              printf("XXXChoice: ");
              scanf("%d",&w);
               printf("");
            }
        d2 = 2;
        break;
    }
  }
  fprintf(f1,"%2");
  erase_text();
  return;
}

technolo_menu()
{
   printf("");
   printf("SEC1.");
```

```
    menu_lines(24);
    printf("XXXTECHNOLOGY CHOICES");
    printf("");
    menu_lines(26);
    printf("XXX1. TTL");
    printf("");
    menu_lines(28);
    printf("XXX2. ECL");
    printf("");
    menu_lines(30);
    printf("XXX3. I²L");
    printf("");
    printf("MAKE A.");
    return;
}



constants(f1)

int *f1;

{

    char a;

    erase_text();
    meaning("CONSTANTS");
    printf("");
    printf "SEC7.";
    find_line(25);
    printf("XXXnot yet implemented");
    printf " type 'r' to return ");
    printf("control to main program");
```

```
    printf("");
    while((a = getchar()) != 'r')
      ; /* wait */
    erase_text();
    return;
  }


codes_text()
  {
    printf("");
    printf("S=C15.");
    printf("MAR9.E=2.");
    find_line(5);
    printf("%4XThe codes used may be standard");
    printf(" codes such as ASCII or they ");
    printf("may be defined by the designer.");
    printf(" Each code is identified ");
    printf("by an id name. User defined codes");
    printf(" are composed of a binary ");
    printf("precision followed by symbols and");
    printf(" values. Standard Codes are identified ");
    printf("by an id name and the standard code name. ");
    printf("Please make the desired choices from the");
    printf(" list at the right.");
    printf("");
    printf("MAR=.2.");
    return;
  }

codes_menu1()
  {
    printf("");
    printf("S=C1." );
```

```
    menu_lines(1);
    printf("MARGOO.757.");
    printf("XXXCodes may be ");
    printf("classified as one of ");
    printf("the following: ");
    printf("");
    menu_lines(5);
    printf("XXX0. EMPTY");
    printf("");
    menu_lines(8);
    printf("XXX1. CODE ");
    printf("");
    menu_lines(11);
    printf("XXX2. CODE ");
    printf("    VARIABLES");
    printf("");
    menu_lines(14);
    printf("XXX3. FINISHED");
    printf("");
    menu_lines(20);
    printf("XXXMake a choice by typing ");
    printf("the number by the ");
    printf("desired category.");
    printf("");
    printf("MARK.2.");
    find_line(25);
    printf("XXXChoice: ");
    return;
}

codes_menu2()
{
    printf("");
    printf("SEC1.");
```

```
    menu_lines(1);
    printf("MAR85.757.");
    printf("XXXValid choices are:");
    printf("now the following: ");
    printf("");
    menu_lines(5);
    printf("XXX1. CODE 1");
    printf("");
    menu_lines(6);
    printf("XXX2. CODE 2");
    printf("    VARIABLES");
    printf("");
    menu_lines(11);
    printf("XXX3. FINISHED");
    printf("");
    menu_lines(15);
    printf("XXXMake a choice by typing:");
    printf("the number by the ");
    printf("desired category.");
    printf("");
    printf("MAR2.4.");
    find_line(25);
    printf("XXXChoice: ");
    return;
  }

codes(f1)

  int *f1;

  {

    int v, d2;
```

```
erase_text();
heading("CODE PARAMETERS");
codes_text();
codes_menu1();
scanf("%d",&v);
printf("");
if(v == 0)
 {
  erase_text();
 }
else
 {
  d2 = 0;
  printf("");
  while(d2 == 0)
   {
    switch (v)
     {
      case 1:
       fprintf(f1,"   CODE : 0);
       user_code_parameters(f1);
       heading("CODE PARAMETERS");
       codes_text();
       codes_menu2();
       d2 = 0;
       scanf("%d",&v);
       printf("");
       break;
      case 2:
       fprintf(f1,"   CODE VARIABLES : 0);
       code_vars_parameters(f1);
       heading("CODE PARAMETERS");
       codes_text();
       codes_menu2();
       d2 = 0;
```

```
            scanf("%d",&v);
            printf("");
            break;
          case 3:
            d2 = 1;
            break;
          default:
            while((v < 1) || (v > 3))
              {
                printf("");
                printf("%aC1.");
                find_line(20);
                printf("XXXPlease choose one from the");
                printf("given list.");
                printf("");
                printf("%aC2.");
                find_line(23);
                printf("XXXChoice: %d",v);
                printf("");
                printf("%aC1.");
                find_line(23);
                printf("XXXChoice: ");
                scanf("%d",&v);
                printf("");
              }
            d2 = 0;
            break;
        }
      }
    }
  printf(fl,"4");
  erase_text();
  return;
}
```

```
user_code_text()
 {
  printf("");
  printf("SEC16.");
  printf("MAR9.552.");
  find_line(5);
  printf("XXXThe codes entered in this");
  printf(" section are defined by the ø,;
  printf("designer.Each code is identified");
  printf(" by an id name. User defined codes");
  printf(" are composed of a binary ø);
  printf("precision followed by symbols and");
  printf(" values. Ø,;
  printf("Please make the desired choices from the");
  printf(" list at the right.");
  printf("");
  printf("MAR9.9.");
  return;
 }


user_code_menu1()
 {
  printf("");
  printf("SEC1.");;
  menu_lines(1);
  printf("MAR55.757.");
  printf("XXXUser code ø);
  printf("parameters may be ø);
  printf("entered using the ø);
  printf("following: ø);
  printf("");
  menu_lines 7.;
  printf("XXXø. NONE");
  printf("");
```

```
    menu_lines(13);
    printf("XXX1. ENTER USER CODE #");
    printf("");
    menu_lines(14);
    printf("XXX2. FINISHED");
    printf("");
    menu_lines(18);
    printf("XXXMake a choice by typing #");
    printf("the number by the #");
    printf("desired category.");
    printf("");
    printf("MAIN.#.");
    find_line(25);
    printf("XXXChoice: ");
    return;
}


user_code_menu2()
{
    printf("");
    printf("SEC1.");
    menu_lines(1);
    printf("MAREC.757.");
    printf("XXXValid choices #");
    printf("now are: #");
    printf("");
    menu_lines(7);
    printf("XXX1. ENTER USER CODE #");
    printf("");
    menu_lines(10);
    printf("XXX2. FINISHED");
    printf("");
    menu_lines(14);
    printf("XXXMake a choice by typing #");
```

```
    printf("the number by the ");
    printf("desired category.");
    printf("");
    printf("MARC.0.");
    find_line(22);
    printf("XXXChoice: ");
    return;
   }


user_code_parameters(x1)

 int *f1;

 {

  char c, id[24], id_char[20];
  int w, x, y, d3, d4;
  int value;

  erase_text();
  heading("USER CODE PARAMETERS");
  user_code_text();
  user_code_menu1();
  scanf("%d",&w);
  printf("");
  if(w == 0)
   {
    erase_text();
   }
  else
   {
    d3 = 0;
    printf("");
    while(d3 == 0)
```

```
{
 switch (w)
  {
   case 1:
    printf("");
    find_line(15);
    printf("XXXCODE ID: ");
    c = getchar();
    getline(id);
    fprintf(f1,"      %s , ",id);
    printf("");
    find_line(17);
    printf("XXXBINARY PRECISION: ");
    scanf("%d",&y);
    fprintf(f1,"%d , ",y);
    printf("");
    printf("SEC0.");
    find_line(25);
    printf("XXXChoice: %c",w);
    printf("");
    printf("SEC1.");
    find_line(21);
    printf("XXXChoose from the list ");
    printf("at the right.");
    character_menu();
    printf("");
    find_line(22);
    printf("XXXChoice: ");
    scanf("%d",&x);
    d4 = 2;
    while(d4 == 3)
     {
      switch (x)
       {
        case 1:
```

```
printf("");
find_line(24);
printf("XXXCHARACTER ID: ");
c = getchar();
getline(id_char);
fprintf(f1,"%s : ",id_char);
printf("");
printf("SEC1.");
find_line(26);
printf("XXXCODE NUMBER: ");
scanf("%d",&value);
fprintf(f1,"%d ; ",value);
printf("");
printf("SEC2.");
find_line(24);
printf("XXXCHARACTER ID: %s",id_char);
find_line(26);
printf("XXXCODE NUMBER: %d",value);
printf("");
printf("SEC0.");
find_line(22);
printf("XXXChoice: %d",x);
printf("");
printf("SEC1.");
find_line(22);
printf("XXXChoice: ");
scanf("%d",&x);
d4 = x;
break;
case 2:
d4 = 1;
break;
default:
while((x < 1) || (x > 2))
 {
```

```
        printf("\n");
        printf("SAC1.");
        find_line(22);
        printf("XXXPlease choose one from the");
        printf("given list.");
        printf("\n");
        printf("SEC%.");
        find_line(22);
        printf("XXX      %d",x);
        scanf("%d",&x);
        printf("\n");
        printf("SEC%.",
        find_line(22);
        printf("XXXPlease choose one from the");
        printf("given list.");
        printf("\n");
        printf("SAC1.");
      }
      if=0;
      break;
    }
  }
  fprintf(11,"%d",x);
  erase_text();
  reading USER CODE PARAMETERS");
  user_code_text();
  user_code_menu();
  do {x=0;
  scanf("%d",&x);
  printf("\n");
  break;
  case 2:
  fprintf(11,"                    END CODE\n");
  do - 1;
  break;
```

```
        default:
         while( w < 1 || (w > 2)
          {
           printf("");
           printf("SEC1.");
           find_line(24);
           printf("XXXPlease choose one from the");
           printf(" given list.");
           printf("");
           printf("SEC4.");
           find_line(25);
           printf("XXXChoice: %d",w);
           printf("");
           printf("SEC1.");
           find_line(25);
           printf("XXXChoice: ");
           scanf("%d",&w);
           printf("");
           printf("SEC4.");
           find_line(24);
           printf("XXXPlease choose one from the");
           printf(" given list.");
           printf("");
           printf("SEC1.");
          }
         d2 = w;
         break;
       }
     }
   }
  fprintf(f1,"4");
  erase_text();
  return;
}
```

```
character_menu()
 {
  printf("");
  printf("SEC1.");
  menu_lines(24);
  printf("XXXOPTIONS");
  printf("");
  menu_lines(26);
  printf("XXX1. ENTER CHARACTER");
  printf("");
  menu_lines(27);
  printf("    REPRESENTATION");
  printf("");
  menu_lines(30);
  printf("XXX2. FINISHED");
  printf("");
  printf("MARK.0.");
  return;
 }


code_vars_text()
 {
  printf("");
  printf("SEC15.");
  printf("MARK.SEC.");
  find_line(5);
  printf("XXXEach code_variable is to be");
  printf(" specified by an identifier,");
  printf("and the type of standard code");
  printf(" used with that identifier.");
  printf("The identifiers used to name the");
  printf(" codes must begin with a letter");
  printf("and should be no longer than");
```

```
    printf(" eight characters. @");
    printf("Please enter the appropriate information");
    printf(" as it is requested. @");
    printf("Please follow the instructions listed");
    printf(" to the right. @");
    printf("");
    printf("MARK.@.");
    return;
  }


code_vars_menu1()
  {
    printf("");
    printf("SEC1.");
    menu_lines(1);
    printf("MAR505.757.");
    printf("XXXCode variable @ ;
    printf("parameters may be @");
    printf("entered using the @");
    printf("following: @");
    printf("");
    menu_lines(7);
    printf("XXXV. NONE");
    printf("");
    menu_lines(14);
    printf("XXX1. ENTER CODE VARIABLE @");
    printf("    PARAMETER");
    printf("");
    menu_lines(14);
    printf("XXX2. FINISHED");
    printf("");
    menu_lines(16);
    printf("XXXMake a choice by typing ...;
    printf(" the number by the @");
```

```
    printf("desired category.");
    printf("");
    printf("MARK.#.");
    find_line(25);
    printf("XXXChoice: ");
    return;
}


code_vars menu2()
{
    printf("");
    printf("SEC1.");
    menu_lines(1);
    printf("VAR565.757.");
    printf("XXXValid choices #");
    printf("now are: #");
    printf("");
    menu_lines(7);
    printf("AXX1. ENTER CODE VARIABLE #");
    printf("    PARAMETER");
    printf("");
    menu_lines(10);
    printf("XXX2. FINISHED");
    printf("");
    menu_lines(14);
    printf("XXXMake a choice by typing #");
    printf("the number by the #");
    printf("desired category.");
    printf("");
    printf("MARK.#.");
    find_line(25);
    printf("XXXChoice: ");
    return;
}
```

```
code vars parameters(f1)

 int *f1;

 {

  char c, id[20];
  int w, x, y, d2, d4;

  erase_text();
  headin("CODE VARIABLE PARAMETERS");
  code_vars_text();
  code_vars_menu1();
  scanf("%d",&w);
  printf("");
  if(w == 0)
   {
    erase_text();
   }
  else
   {
    d2 = 0;
    printf("");
    while(d2 == 0)
     {
      switch (w)
       {
        case 1:
         printf("");
         find_line(15);
         printf("XXXCODE ID: ");
         c = getchar();
         getline(id);
         fprintf(f1,"      %c : ",id );
         printf("");
```

```
standard_code_menu();
printf("");
printf("SEC0.");
find_line(25);
printf("XXXChoice: %d",w);
printf("");
printf("SEC1.");
find_line(18);
printf("XXXChoose from the list ");
printf("at the right.");
printf("");
find_line(20);
printf("XXXCOLD: ");
scanf("%d",&x);
d4 = 0;
while(d4 == 0)
 {
  switch (x)
   {
    case 1:
     fprintf(f1," ASCII6 ;");
     d4 = 1;
     break;
    case 2:
     fprintf(f1," ASCII7 ;");
     d4 = 1;
     break;
    case 3:
     fprintf(f1," BCDIC ;");
     d4 = 1;
     break;
    case 4:
     fprintf(f1," LCL ;" ;
     d4 = 1;
     break;
```

```
      default:
       while((x < 1) || (x > 4))
        {
         printf("");
         printf("SEC1.");
         find_line(28);
         printf("XXXPlease choose one from the");
         printf("given list.");
         printf("");
         printf("SEC0.");
         find_line(29);
         printf("XXX      %d",x);
         printf("");
         printf("SEC1.");
         scanf("%d",&x);
         printf("");
        }
       d4 = 0;
       break;
     }
    printf("");
    printf("SEC1.");
    find_line(28);
    printf("XXXChoice: ");
   }
 fprintf(f1," 0");
 erase_text();
 heading("CODE VARIABLE PARAMETERS");
 code_vars_text();
 code_vars_menu2();
 d3 = 0;
 scanf("%d",&x);
 printf("");
 break;
case 2:
```

```
        fprintf(f1,"    END CODE VARIABLES");
        d3 = 1;
        break;
       default:
        while((w < 1) || (w > 4))
         {
          printf("");
          printf("SEC1.");
          find_line(20);
          printf("XXXPlease choose one from the");
          printf("given list.");
          printf("");
          printf("SECd.");
          find_line(25);
          printf("XXXChoice: %d",w);
          printf("");
          printf("SEC1.");
          find_line(23);
          printf("XXXChoice: ");
          scanf("%d",&w);
          printf("");
         }
        d3 = 3;
        break;
      }
    }
  }
 fprintf(f1,"");
 erase_text();
 return;
}
```

```
standard code_menu()
 {
  printf("");
  printf("SAC1.");
  menu_lines(24);
  printf("XXXCODE CHOICES");
  printf("");
  menu_lines(26);
  printf("XXX1. ASCII6");
  printf("");
  menu_lines(28);
  printf("XXX2. ASCII7");
  printf("");
  menu_lines(30);
  printf("XXX3. EBCDIC");
  printf("");
  menu_lines(32);
  printf("XXX4. BCD");
  printf("");
  printf("MARK.");
  return;
 }

/*##########################################################
###  End Environment Section                             ###
##########################################################*/


/*##########################################################
### These functions handle the Contingency               ###
### List Section of a design in CSDL.                    ###
##########################################################*/
```

```
contingency_section(f1)

int *f1;

{

  char c;

  erase_text();
  heading("CONTINGENCY LIST SECTION");
  printf("\n");
  printf("SEC7.");
  find_line(25);
  printf("XXXnot yet implemented");
  printf("  type 'r' to return ");
  printf("control to main program");
  printf("\n");
  while((c = getchar()) != 'r')
     ; /* wait */
  erase_text();
  return;
}


/*##########################################################
###  End Contingency List Section                        ###
##########################################################*/


/*##########################################################
### These functions handle the Procedures                ###
### Section of a design in OSDL.                         ###
##########################################################*/

procedures_section(f1,
```

```
int *f1;

{

 char a;

 erase_text();
 heading("PROCEDURES SECTION");
 printf("");
 printf "SEC?.",;
 find_line(25);
 printf("XXAnot yet implemented");
 printf(" type 'r' to return 0);
 printf("control to main program ");
 printf("");
 while( a = getchar() != 'r'
    ; /* wait */
 erase_text();
 return;
}

/*##########################################################
###  End Procedures Section                               ###
##########################################################*/

/*##########################################################
###    This section contains the graphics utilit,        ###
###    functions used by the other sections.             ###
##########################################################*/
```

```
/* This block of code draws a box around the screen */

boundry()
 {
  printf("");
  printf("ESC");
  printf("");
  printf("SEC128.MOV4.0.");
  printf("DVA0.575.DVA767.575.MOV767.574.");
  printf("DVA0.574.DVA767.574.");
  printf("DVA767.0.DVA0.0.MOV0.1.");
  printf("DVA767.1.MOV0.100.");
  printf("DVA767.100.MOV767.101.DVA0.101.");
  return;
 }


/* This block draws the work area screen */

workarea()
 {
  boundry();
  printf("SEC128.");
  printf("MOV4.535.DVA560.535.");
  printf("MOV560.534.DVA0.534.");
  printf("MOV560.575.DVA560.100.");
  return;
 }


/* This block of code fprintfs the user
   prompts required for this
   screen in the user prompt area.        */
```

```
prompt1()
 {
  printf("");
  printf("SEC2.");
  find_line(23);
  printf("XXXType 's' to Start");
  printf("");
  find_line(23);
  printf("XXXType 'q' to Quit");
  printf("");
  return;
 }


prompt2()
 {
  printf("");
  printf("SEC2.");
  find_line(23);
  printf("XXXType 'i' to enter id section");
  printf("");
  find_line(23);
  printf("XXXType 'q' to Quit");
  printf("");
  return;
 }


prompt3()
 {
  printf("");
  printf("SEC2.");
  find_line(23);
  printf("XXXType 'q' to Quit");
```

```
    printf("");
    find_line(38);
    printf("XXXType 'd' to Enter Design Criteria");
    printf("");
    find_line(40);
    printf("XXXType 'e' to Enter Environment Section");
    printf("");
    find_line(41);
    printf("XXXType 'c' to Enter Contingency List Section");
    printf("");
    find_line(42);
    printf("XXXType 'p' to Enter Procedures Section");
    printf("");
    return;
}


prompt4()
{
    printf("");
    printf("Shoz.");
    find_line(38);
    printf("XXXType 'q' to Quit");
    printf("");
    find_line(39);
    printf("XXXType 'e' to enter Environment Section");
    printf("");
    find_line(40);
    printf("XXXType 'c' to enter Contingency List Section");
    printf("");
    find_line(41);
    printf("XXXType 'p' to enter Procedures Section");
    printf("");
    return;
}
```

```
prompt5(
 {
  printf("");
  printf("SLC2.");
  find_line(38);
  printf("XXXType 'q' to Quit");
  printf("");
  find_line(39);
  printf("XXXType 'c' to Enter Contingency List Section");
  printf("");
  find_line(40);
  printf("XXXType 'p' to Enter Procedures Section");
  printf("");
  return;
 }


prompt6(
 {
  printf("");
  printf("SLC2.");
  find_line(38);
  printf("XXXType 'q' to Quit");
  printf("");
  find_line(40);
  printf("XXXType 'p' to Enter Procedures Section");
  printf("");
  return;
 }


prompt7(
 {
  printf("");
  printf("SLC2.");
```

```
    find_line(38.);
    printf("AXXType 'q' to Quit");
    printf("");
    return;
  }

heading(text)
 char text[];
 {
  printf("");
  printf("SEG".SAP272.30.L");
  printf("MOVE0.540.");
  printf("XXX%s",text);
  printf("");
  printf("SAP178.13.L");
 }


erase_text()
 {
  printf("");
  printf("SEG4.");
  printf("SWM127.");
  printf("RES");
  printf("SWM255.");
  return;
 }

erase_menu()
 {
  printf("");
  printf("SEG4.");
  printf("SWM1.");
  printf("RES");
  printf("SWM255.");
```

```
    return;
  }

getline(s)
 char s[];
 {
  char c;
  int i;

  for(i=0; i<MAXLINE && (c=getchar()) != '\n';++i
   {
    s[i] = c;
   }
  s[i] = '\0';
  return;
 }


find_line(i)
 int i;
 {
  printf("");
  switch (i)
   {
    case 1:
     printf("MOV3.561.");
     break;
    case 2:
     printf("MOV3.564.");
     break;
    case 3:
     printf("MOV3.535.");
     break;
    case 4:
```

```
printf("MOV=.324.");
break;
case 5:
printf("MOV=.32=.");
break;
case 6:
printf("MOV=.490.");
break;
case 7:
printf("MOV=.4b5.");
break;
case 8:
printf("MOV=.472.");
break;
case 9:
printf("MOV=.457.");
break;
case 10:
printf("MOV=.442.");
break;
case 11:
printf("MOV=.431.");
break;
case 12:
printf("MOV=.41?.");
break;
case 13:
printf("MOV=.4??.");
break;
case 14:
printf("MOV=.3?4.");
break;
case 15:
printf("MOV=.???.");
break;
```

```
case 16:
 printf("MOV9.366.");
 break;
case 17:
 printf("MOV9.353.");
 break;
case 18:
 printf("MOV9.340.");
 break;
case 19:
 printf("MOV9.327.");
 break;
case 20:
 printf("MOV9.314.");
 break;
case 21:
 printf("MOV9.301.");
 break;
case 22:
 printf("MOV9.288.");
 break;
case 23:
 printf("MOV9.275.");
 break;
case 24:
 printf("MOV9.262.");
 break;
case 25:
 printf("MOV9.249.");
 break;
case 26:
 printf("MOV9.236.");
 break;
case 27:
 printf("MOV9.223.");
```

```
        break;
      case 20:
        printf("MOV 2,21.");
        break;
      case 21:
        printf("MOV 2,137.");
        break;
      case 22:
        printf("MOV 2,164.");
        break;
      case 31:
        printf("MOV 2,171.");
        break;
      case 32:
        printf("MOV 2,126.");
        break;
      case 33:
        printf("MOV 2,142.");
        break;
      case 34:
        printf("MOV 2,152.");
        break;
      case 35:
        printf("MOV 2,113.");
        break;
      case 26:
        printf("MOV 2,146.");
        break;
      case 27:
        printf("MOV 2,[illegible].");
        break;
      case 24:
        printf("[illegible].");
        break;
      case [illegible]:
```

```
        printf("MOVE.57.");
        break;
      case 40:
        printf("MOVE.54.");
        break;
      case 41:
        printf("MOVE.41.");
        break;
      case 42:
        printf("MOVE.28.");
        break;
      case 43:
        printf("MOVE.15.");
        break;
      case 44:
        printf("MOVE.2.");
        break;
      default:
        break;
    }
  return;
}

send_lines(i)
 int i;
 {
  printf "";
  switch (i)
   {
    case 1:
      printf("MOVE.546.");
      break;
    case 2:
      printf("MOVE.
```

```
break;
case 5:
 printf("NOV365.5%f.");
break;
case 4:
 printf("NOV330.5%3.");
break;
case 3:
 printf("NOV303.4%3.");
break;
case 2:
 printf("NOV333.4%7.");
break;
case 1:
 printf("NOV444.");
break;
case 0:
 printf("NOV365.4%3.");
break;
case 7:
 printf("NOV333.42%.");
break;
case 8:
 printf("NOV355.4%7.");
break;
case 9:
 printf("NOV333.4%5.");
break;
case 10:
 printf("NOV333.4%5.");
break;
case 11:
 printf("NOV333.4%5.");
break;
case 12:
 printf("NOV333.4%5.");
break;
case 13:
 printf("NOV333.4%5.");
break;
case 14:
```

```
printf("Novice.\n");
break;
case 10:
printf("Novice.\n");
break;
case 16:
printf("Novice.\n");
break;
case 17:
printf("Novice.\n");
break;
case 18:
printf("Novice.\n");
break;
case 19:
printf("Novice.\n");
break;
case 20:
printf("Novice.\n");
break;
case 21:
printf("Novice.\n");
break;
case 22:
printf("Novice.\n");
break;
case 23:
printf("Novice.\n");
break;
case 24:
printf("Novice.\n");
break;
case 25:
printf("Novice.\n");
break;
```

```
case 26:printf("MOV$05.223.");
        break;
case 27:printf("MOV$03.218.");
        break;
case 28:printf("MOV$05.187.");
        break;
case 29:printf("MOV$05.184.");
        break;
case 30:printf("MOV$05.171.");
        break;
case 31:printf("MOV$05.146.");
        break;
case 32:printf("MOV$05.141.");
        break;
case 33:printf("MOV$05.132.");
        break;
case 34:printf("MOV$05.113.");
        break;
case 35:printf("MOV$05.11?.");
        break;
default:
        exit();
}
return;
```

APPENDIX C

DESIGN EXAMPLE

```
IDENTIFICATION
  DESIGNER : Duard Stephen Woffinden
  DATE : 19 June 1984
  PROJECT : Thesis Demonstration

DESIGN CRITERIA
  METRIC POWER ;
  VOLUMES 2, 1, 3 ;
  MONITORS 1, 2, 3 ;

ENVIRONMENT
  OUTPUT :
    out1 , 8 , TTL ;
    out2 , 8 , TTL ;
    out3 , 16 , ECL ;
  END OUTPUT

  INPUT :
    in1 , 2 , ITL ;
    in2 , 8 , TTL ;
    in3 , 8 , ITL ;
  END INPUT

  DUPLEX :
    in-out1 , 8 , ITL ;
    out-in1 , 8 , ECL ;
  END DUPLEX

  CODE :
    code 1 , 8 , A : 25 ; B : 26 ;
    code 2 , 8 , A : 0 ; B : 1 ; C : 2 ;

  CODE VARIABLES :
    code3 : ASCII7 ;
    code4 : EBCDIC ;
    code5 : BCD ;
  END CODE VARIABLES
```

LIST OF REFERENCES

1. Lawrence Livermore Laboratory Report VCRL-78651, Automating the Design of Dedicated Real Time Control Systems, by M.N. Matelan, 21 August 1976.

2. Ross, A.A., Computer Aided Design of Microprocessor-Based Controllers, Ph.D. Thesis, University of California, Davis, 1978.

3. Sherlock, B.J., User-Friendly Syntax Directed Input to a Computer Aided Design System, M.S. Thesis, Naval Postgraduate School, Monterey, California, 1983.

4. Carson, T.H., Input Translator for a Computer Aided Design System, M.S. Thesis, Naval Postgraduate School, Monterey, California, 1984.

5. Licklider, J.C.R., "Man-Computer Symbiosis," IEEE Professional Group on Human Factors Engineering, 18 January 1960.

6. Foley, J.D., Van Dam, A., Fundamentals of Interactive Graphics, Addison-Wesley Publishing Company, March 1983.

7. Bailey, R.W., Human Performance Engineering: A Guide for System Designers, Prentice-Hall, Inc., New Jersey 1982.

8. Foley, J.D., Wallace, V.L., "The Art of Natural Graphic Man-Machine Conversation," Proceedings of the IEEE, April 1979.

9. Jones, P.F., "Four Principles of Man-Computer Dialogue," Computer-Aided Design, Vol. 10, No. 3, IPC Science and Technology Press Limited, Surrey, England, May 1978.

10. Norman, D.A., "Design Rules Based on Analysis of Human Error," Communications of the ACM, Vol. 26, No. 4, April 1983.

11. Hansen, W.J., "User Engineering Principles for Interactive Systems," AFIPS, Fall Joint Computer Conference Proceedings, Vol. 39, 1971, Interactive Programming Environments, pp. 217-231, McGraw-Hill, 1984.

12. Martin, J., Design of Man-Computer Dialogues, Prentice Hall, Inc., Englewood Cliffs, N.J., 1973.

13. Card, S.K., Moran, T.P., Newell, A., The Psychology of Human-Computer Interaction, Lawrence Erlbaum Association Publishers, 1983.

14. Lehman, M.M., "The Environment of Program Development and Maintenance--Programs, Programming, and Programming Support," Proceedings, 1981 International Computing Symposium, pp. 1-12, IPC Business Press, Ltd., 1981.

15. Groenert, F.E. Jr., The Design Analysis and Implementation of a User-Friendly Interface for the Unix Operating System, M.S. Thesis, Naval Postgraduate School, Monterey, California, 1984.

16. Winograd, T., "Breaking the Complexity Barrier (Again)," Proceedings of the ACM SIGPLAN-SIGIR Interface Meeting on Programming Languages--Information Retrieval, November 1973, Interactive Programming Environments, pp. 3-18, McGraw-Hill, 1984.

17. Sheil, B.A., "Power Tools for Programmers," Datamation Magazine, Technical Publishing Company, 1983, Interactive Programming Environments, pp. 19-30, McGraw-Hill, 1984.

18. Barstow, D.R., Shrobe, H.E., Sandewall, E., Interactive Programming Environments, McGraw-Hill, 1984.

19. AED User's Manual, Advanced Electronics Design, Inc., Sunnyvale, California, April 1983.

20. Unix Programmer's Manual, Computer Science Division, University of California, Berkeley, California, seventh Edition, June 1981.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943	2
4.	Curriculum Officer, Code 37 Computer Technology Naval Postgraduate School Monterey, California 93943	1
5.	Professor Alan A. Ross, Code 52Rs Computer Science Department Naval Postgraduate School Monterey, California 93943	3
6.	Professor George A. Rahe, Code 52Ra Computer Science Department Naval Postgraduate School Monterey, California 93943	1
7.	Duard S. Woffinden 496 Canyon Road Logan, Utah 84321	4

210841

Thesis
W6864 Woffinden
c.1 An interactive environment for a computer-aided design system.